International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(20s) 2026 pp. 1464-1475
https://doi.org/10.70917/ijcisim-2026-5997
Article

A Cloud-Native Reference Architecture for Governed Observability and Explainable Operational Intelligence

Sai Dheeraj Guntupalli

Independent Researcher, USA. ORCID: 0009-0004-2503-1134

Abstract: Enterprise observability today is assembled from disconnected tools for collection, storage, visualization, and automated operations, leaving governance, lineage, and explainability fragmented across the telemetry lifecycle. This paper presents a cloud-native reference architecture that treats operational telemetry as a governed enterprise asset rather than a byproduct of individual tools. The architecture specifies layered telemetry acquisition, event-driven processing, a governed telemetry lakehouse, and an Operational Intelligence Engine that produces anomaly detection, incident correlation, root cause support, and forecasting outputs, each accompanied by an auditable explanation artifact rather than an opaque recommendation. Governance, meaning catalog, schema, lineage, and policy, is embedded directly into the processing path rather than layered on afterward, and high-impact automated actions default to human approval. The architecture is evaluated through a methodology intended for implementers, including a disclosed simulation study that illustrates how governance-aware correlation degrades gracefully under incomplete metadata where rule-based approaches degrade sharply. The contribution is systems integration and governance contracts for trustworthy operational intelligence, not a new instrumentation standard or a single detection model. Enterprises adopting this architecture can expect clearer accountability for automated operational decisions, improved ability to reconstruct incident history, and a practical path toward observability practices that satisfy both operational and audit requirements as cloud-native estates continue to grow in scale and complexity.

Keywords: Cloud-Native Observability, AIOps, Metadata Governance, Telemetry Lakehouse, Explainable Artificial Intelligence, Distributed Systems Architecture

1. Introduction

A. Motivation

Enterprise applications increasingly run as independently deployable services distributed across container orchestration clusters, hybrid clouds, and edge sites, a decomposition that trades a single monolithic failure domain for a much larger surface of inter-service calls that must themselves be observed [15]. Every layer of this stack emits telemetry. The three pillars of observability, namely metrics, logs, and traces, remain foundational to how operators reason about system behavior, and open instrumentation standards have meaningfully improved vendor-neutral collection of that telemetry [3], [4], [5]. In parallel, a distinct research thread applies machine learning to anomaly detection and incident correlation under the label of AIOps [6], [7], while data platform teams have independently matured catalogs, schema registries, and lakehouse storage for analytical workloads [8], [9].

These three threads rarely meet inside a single architectural contract. Production operations still require a human to stitch together signals scattered across disconnected collection, storage, and automation systems. Telemetry schemas drift silently because no contract enforces them. Machine-generated recommendations arrive without provenance, so an operator cannot tell which inputs produced a given alert or how confident the system should be in

it. Retention and access policies get applied inconsistently, if at all, and the observability plane itself is rarely held to the same governance rigor that the organization would demand of its customer-facing data platforms.

B. Positioning and Contribution

This gap is not a tooling problem that a better collector solves, nor is it a claim that existing commercial observability products lack value. The contribution here is a reference architecture that elevates the full telemetry lifecycle, from acquisition through governed automation, under one set of platform contracts, so that governance and explainability stop being bolted onto AIOps after the fact and instead become architectural defaults.

The work makes five contributions. First, it specifies a seven-layer cloud-native reference architecture that unifies telemetry acquisition, streaming, lakehouse storage, governance, AI-driven operational intelligence, visualization, and policy-aware automation inside one contract, where prior proposals typically address only one or two of these layers in isolation. Second, it introduces a telemetry-as-asset model in which catalogued semantics, versioned schemas, lineage, and quality scores travel with the telemetry itself, rather than being reconstructed after the fact by a downstream consumer. Third, it defines an Operational Intelligence Engine contract that requires explainability, governance-aware inference, and human-in-the-loop controls for any high-impact action, addressing the common failure mode in which AIOps deployments optimize predictive accuracy while leaving feature provenance and policy constraints unspecified. Fourth, it specifies a distributed-systems processing model, combining event sourcing, command-query responsibility segregation, and hybrid consistency, that is purpose-built for telemetry workloads rather than adapted from generic microservice patterns [14]. Fifth, it provides an evaluation methodology for implementers, together with an illustrative simulation constructed for architectural validation, without presenting unpublished multi-tenant production benchmarks as completed experimental results.

Where identity-resolution platforms optimize entity matching and consent-bound profile graphs, this architecture optimizes operational signal correlation and closed-loop reliability. The two problem domains share infrastructure patterns such as container orchestration, event buses, and metadata catalogs because those patterns are simply good cloud-native practice, not because the domains themselves overlap; the metamodel, intelligence tasks, and success metrics addressed here are specific to observability and AIOps and should be evaluated on those terms.

C. Organization

Section II surveys related work across observability, instrumentation, AIOps, platform engineering, and data governance, and names the specific integration gap this architecture fills. Section III states the problem and the research objectives. Section IV specifies functional and non-functional requirements together with the design principles that constrain every layer. Section V presents the seven-layer reference architecture. Section VI details the Operational Intelligence Engine. Section VII covers the distributed-systems treatment of telemetry processing. Section VIII provides an evaluation methodology together with a disclosed simulation study. Section IX discusses comparative positioning and limitations, while Section X provides the conclusion.

2. Related Work

A. From Monitoring to Observability

Classical monitoring emphasized host-level metrics and threshold alerts. Application performance monitoring extended the practice with transaction traces and dependency views, and observability further extends the practice by allowing an operator to interrogate previously unknown failure modes using high-cardinality telemetry rather than a fixed dashboard [4]. Distributed tracing systems inspired by early large-scale infrastructure work reconstruct cross-service request paths, giving operators a causal view that host metrics alone cannot provide [10].

B. Open Instrumentation and Cloud-Native Tooling

Open instrumentation standards have standardized the APIs, SDKs, and collectors used for metrics, logs, and traces, substantially reducing vendor lock-in at the collection layer [5]. A broad ecosystem of time-series databases,

visualization layers, and distributed tracing backends now provides storage and rendering for this telemetry [16]. These tools are necessary building blocks, and the architecture proposed here assumes that such components may realize individual layers of the seven-layer model. Its focus sits one level up: governance, correlation contracts, and explainable intelligence that span whichever specific tools an implementer selects.

C. AIOps

Surveys of AIOps describe anomaly detection, event correlation, causal analysis, and ticket automation as the dominant application areas [6], [7], and a growing empirical literature examines how AIOps projects are actually built and maintained in open-source settings, with recent work finding that such projects carry a higher defect burden than comparable machine learning projects of similar scope [21]. Many production deployments in this space optimize predictive accuracy while leaving feature provenance, model versioning, and policy constraints on automated remediation under-specified. This architecture treats those concerns as mandatory defaults rather than optional hardening.

D. Platform Engineering

Internal developer platforms, GitOps delivery, and service meshes have improved delivery velocity and operational consistency across large engineering organizations [1], [2], [11]. The cloud-native ecosystem itself has converged on a shared vocabulary and landscape of container orchestration, service mesh, and observability tooling that platform teams now treat as a common baseline rather than a differentiator [17], and empirical research on software delivery performance consistently links high deployment frequency and short lead times to the same platform investments [18]. Within most of these platforms, observability is a capability the platform consumes rather than a data product the platform governs. This architecture inverts that relationship, treating the observability plane itself as a platform product with its own service-level objectives, schemas, and stewardship responsibilities.

E. Data Governance and Lakehouses

Metadata catalogs, schema registries, and lakehouse storage have matured considerably for analytical data, unifying batch and streaming workloads under a shared governance model [8], [9]. Applying these same patterns to operational telemetry, particularly at the cardinality that modern microservice fleets generate, remains uneven in both the literature and in practice. This architecture makes a telemetry lakehouse and its associated lineage a first-class layer rather than an afterthought bolted onto an existing storage tier.

F. Research Gap

Taken together, prior work advances instrumentation standards, storage backends, visualization, or AIOps models, almost always in isolation from one another. What remains missing is a cohesive reference architecture that defines telemetry lifecycle zones with explicit governance artifacts, binds AIOps outputs to evidence and policy gates rather than leaving them free-floating, and specifies distributed-processing semantics suitable for multi-region cloud-native operation. The architecture presented here addresses that specific integration gap.

3. Problem Statement and Objectives

A. Problem Statement

Enterprises struggle to convert fragmented telemetry into trustworthy operational intelligence precisely because collection, governance, analytics, and automation are architected as separate concerns by separate teams. The research question this work addresses is: how can cloud-native platform engineering, distributed telemetry processing, enterprise metadata governance, and explainable AI-assisted operations be integrated into a single reference architecture that supports scalable and auditable enterprise observability.

B. Objectives

The architecture pursues five objectives. It defines a layered structure for end-to-end telemetry lifecycle management. It embeds catalog, schema, lineage, quality, and retention policy directly into telemetry processing rather than applying them after storage. It specifies an Operational Intelligence Engine contract that requires explanation artifacts and human oversight for any consequential action. It applies event-driven, multi-region processing patterns suitable for high-volume telemetry workloads. It supplies an evaluation methodology covering scalability, resilience, governance, and AIOps quality, so implementers have a concrete path from architecture to evidence.

C. Non-Goals

This architecture does not mandate a single vendor backend for any layer, and no claim is made here of measured superiority over any named commercial observability product. Achieving state-of-the-art results on public anomaly-detection benchmarks is out of scope; instead, the focus is on architectural integrability and governable operational intelligence.

4. Requirements and Design Principles

A. Functional Requirements

The architecture must support multi-source telemetry acquisition spanning applications, container platforms, infrastructure, service mesh, security systems, and business events. It must validate, enrich, correlate, and route streamed telemetry. It must maintain unified storage zones spanning raw, curated, semantic, and analytics tiers. It must provide a metadata catalog, schema registry, lineage tracking, and quality scoring. It must also support AI-assisted anomaly detection, correlation, root-cause support, and forecasting. It must render visualization and service-level-objective views over correlated, governed data. It must gate automation, including alerting, ticketing, scaling, and rollback actions, behind policy evaluation.

Each of these functional requirements maps onto one or more layers in Section V, and none of them stands alone as a sufficient condition for trustworthy operational intelligence. Acquisition without schema stamping produces telemetry that governance cannot later classify. Correlation without lineage produces incident narratives that cannot be reconstructed once the underlying data has aged out of hot storage. Forecasting without quality scoring produces confident projections built on telemetry an operator would never trust if they inspected its provenance directly. Treating these requirements as a single interdependent set, rather than a checklist to satisfy independently, is what distinguishes a governed observability platform from a collection of well-built point tools.

ID	Requirement
FR 1	Multi-source telemetry acquisition (apps, containers, infra, mesh, security, business events)
FR 2	Stream validation, enrichment, correlation, and routing
FR 3	Unified storage zones (raw, curated, semantic, analytics)
FR 4	Metadata catalog, schema registry, lineage, quality scoring
FR 5	AI-assisted anomaly, correlation, RCA support, forecasting
FR 6	Visualization and SLO views over correlated, governed data
FR 7	Policy-gated automation (alert, ticket, scale, rollback)

Table I. Functional Requirements

B. Non-Functional Requirements

The platform must scale under bursty telemetry volume, remain highly available, tolerate faults with replay capability, secure communication through mutual TLS and role- or attribute-based access control, expose the reasoning behind AI outputs, interoperate through open instrumentation standards [5], and observe itself, since a monitoring system that cannot detect its degradation is of limited use during an incident.

C. Principles

Seven principles constrain every layer of the design: cloud-native by design, event-driven processing, governance by design, intelligence by design (meaning AI sits on the processing path rather than in a bolt-on silo), self-observability of the observability platform, security by design, and platform independence, so that the conceptual model remains portable across whichever specific vendor or open-source components an implementer chooses.

5. Reference Architecture

A. Layer Overview

The architecture organizes telemetry lifecycle management into seven layers: sources, acquisition, streaming and processing, telemetry lakehouse, metadata governance, operational intelligence, and visualization and automation.



Figure 1. UTIP seven-layer reference architecture.

Layer	Responsibility
1. Sources	Applications, containers, infrastructure, network, mesh, security, business systems
2. Acquisition	Collectors, agents, SDKs, API gateways; buffer, retry, export
3. Streaming and processing	Validate, enrich, correlate, and route as immutable events
4. Telemetry lakehouse	Raw, curated, semantic, and analytics zones
5. Metadata governance	Catalog, schema, lineage, quality, policies
6. Operational intelligence	Anomaly, correlation, RCA support, forecast, optimize
7. Visualization and automation	Dashboards, SLO views, gated remediation workflows

Table II. Layer Responsibilities

B. Acquisition Layer

Acquisition components perform protocol translation, batching, compression, local buffering, and authenticated export. Instrumentation pipelines compatible with open telemetry standards serve as the preferred default [5], with adapters available for legacy agents where needed. Acquisition must stamp schema version, resource identity, and tenant or environment metadata as early as possible in the pipeline, since governance decisions made downstream depend entirely on metadata that acquisition either captured or permanently lost.

C. Streaming and Event Processing

Telemetry is modeled as a set of immutable events, for example a metric observation, a log line, a completed trace, a successful deployment, or a detected policy violation. Each event passes through validation against the schema registry, enrichment with deployment, ownership, topology, and region context, correlation using trace identifiers, resource identifiers, deployment versions, topology, and time windows, and finally routing to the lakehouse, the intelligence engine, dashboards, automation, or archive according to policy. A distributed log or event bus decouples producers from consumers and enables replay after failure [12].

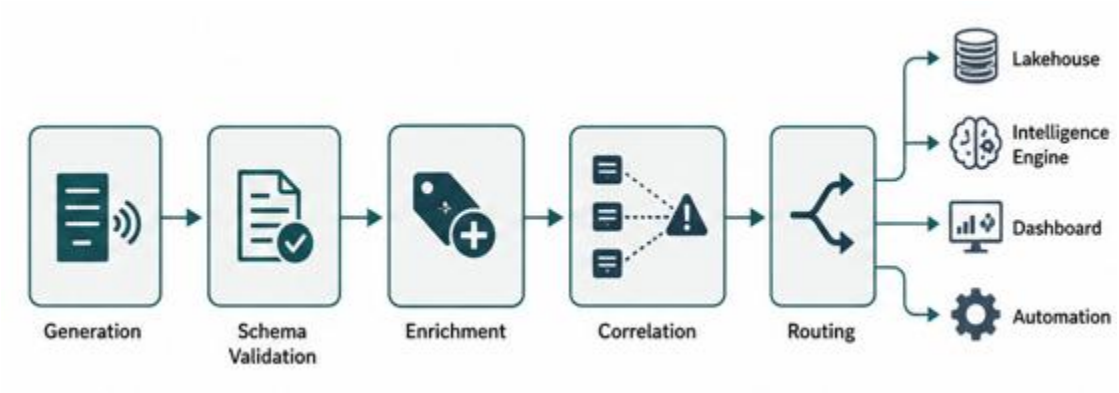


Figure 2. Telemetry event lifecycle from generation through policy-aware automation.

D. Telemetry Lakehouse

Separating storage into raw, curated, semantic, and analytics zones prevents analytical scans from destabilizing the ingest path and preserves the ability to reconstruct historical state for forensics or audit [8], [9].

Zone	Contents	Primary Uses
Raw	Immutable as-received payloads	Forensics, replay, audit
Curated	Normalized, validated, enriched records	Stable operational queries
Semantic	Service graphs, timelines, journey/topology views	Incident context
Analytics	Features, aggregates, ML training sets	Forecasting, reporting

Table III. Telemetry Lakehouse Zones

E. Metadata Governance Layer

The governance layer maintains a catalog recording meaning, ownership, sensitivity, consumers, and service-level-objective relevance for each telemetry stream; a schema registry enforcing compatibility rules as telemetry contracts evolve; lineage tracking; quality scoring across completeness, validity, freshness, and consistency; and a policy repository covering retention, access, residency, sampling, personally identifiable information redaction, and automation privileges. This treatment of redaction and access control as default pipeline behavior, rather than a retrofit applied once a compliance concern is raised, follows the broader principle that privacy protections are most effective

when embedded proactively into system design instead of appended afterward [20]. Governance outputs feed directly into AI confidence scoring and automation gates; they are not documentation produced after the fact.

F. Visualization and Automation

Dashboards and service maps consume curated and semantic data rather than raw telemetry. Automation may open incidents, propose rollbacks, or trigger autoscaling, but only after policy evaluation, and high-impact actions default to requiring human approval, detailed further in Section VI.

G. End-to-End Lifecycle

Across all seven layers, a telemetry event moves from generation at the source, through acquisition and schema stamping, through streaming validation and enrichment, into the lakehouse under governance, and finally into either a human-facing dashboard or a policy-gated automated action, with every step of that path recorded for later audit.

6. Operational Intelligence Engine

The Operational Intelligence Engine is the architecture's primary technical focal point, analogous in importance to confidence scoring inside identity-resolution architectures, but built to solve a distinct correlation and remediation problem rather than an entity-matching one.

A. Capabilities

Anomaly detection, incident correlation, root-cause support, capacity forecasting, and optimization advice make up the engine's core capabilities. It performs anomaly detection, producing scored deviations with evidence windows; incident correlation, grouping related alerts and events; root-cause support, producing ranked hypotheses grounded in topology and time evidence; capacity forecasting, projecting demand and error-budget consumption; and optimization advice, suggesting scaling, placement, or remediation actions. Explainability is treated as a capability in its own right rather than a side effect of the others.

Capability	Output
Anomaly detection	Scored deviations with evidence windows
Incident correlation	Grouped alerts and events into incidents
Root-cause support	Ranked hypotheses with topology/time evidence
Capacity forecasting	Demand and error-budget projections
Optimization advice	Scale, placement, and remediation suggestions
Explainability	Mandatory evidence package per recommendation

Table IV. Operational Intelligence Engine Capabilities

B. Explanation Artifact

Every recommendation the engine produces persists a fixed set of fields: the input telemetry references, including identifiers, time range, and quality scores; the features and topology or deployment context used to generate the recommendation; the model name and version; a confidence or rank value; which policy checks passed or failed; the recommended action together with a blast-radius estimate; and, where applicable, the human decision to approve, reject, or modify the recommendation.

C. Governance-Aware Inference

Before acting on any recommendation, the engine evaluates telemetry quality, lineage availability, schema validity, and applicable automation policy. Missing governance prerequisites reduce the engine's confidence in a

recommendation or block automatic execution outright, so a recommendation built on incomplete or unvalidated telemetry cannot silently masquerade as a high-confidence one.

D. Human-in-the-Loop

The engine is designed to assist operators rather than replace their judgment, a division of labor consistent with established site-reliability practice, which treats error budgets and human on-call judgment as the final arbiter of production risk rather than fully automated decision-making [19]. Operator feedback, whether an acceptance or a rejection, is captured as a governed event and feeds controlled retraining. Where autonomous remediation is enabled at all, it is scoped by explicit policy covering service tier, action type, rate limits, and rollback plan.

E. Continuous Learning Controls

Promoting a model into production requires version pins, validation sets, drift monitors, and lineage links back to the training telemetry that produced it. Historical incidents remain reproducible against the specific model version that generated the contemporaneous recommendation, which matters when an operator later needs to understand why a system behaved as it did during a past incident.

7. Distributed Telemetry Processing

A. Why Distribution Matters

Telemetry is continuous, high-volume, multi-tenant, and geographically partitioned by nature [13]. A centralized, monolithic pipeline becomes both a throughput bottleneck and a single coupled failure domain, which is precisely the opposite of what an observability platform should be during an incident.

B. Patterns

Five patterns recur in the design of distributed processing. Event sourcing maintains a durable, append-only history that supports replay and audit [14]. Command-query responsibility segregation optimizes the write path for high-volume ingest while serving reads through separately optimized models for dashboards and graphs. Hybrid consistency applies strong consistency to schema and policy updates, eventual consistency to aggregates and most dashboards, and a local-first model with asynchronous synchronization in regions experiencing network partitions. Fault tolerance relies on idempotent consumers, retries, dead-letter queues, checkpoints, and replay. Partitioning by service, namespace, tenant, trace, or time window enables both parallelism and isolation between tenants.

C. Multi-Region Operation

Regional clusters ingest and serve local operations independently during a network partition, and metadata, policies, models, and summary analytics synchronize asynchronously once connectivity recovers, so a region cut off from its peers keeps functioning rather than failing closed. This approach mirrors recent orchestration work showing that scheduling and workload placement decisions made independently within each region, rather than routed through a single global scheduler, materially reduce cross-region coordination overhead in multi-region container clusters [22].

This design choice has a direct consequence for governance: a policy update issued from a central control plane during a partition will not reach an isolated region until connectivity is restored, so the architecture must define, in advance, which categories of decision a region is permitted to make using its last-known-good policy state and which categories must instead queue until synchronization resumes. Automation gated on high-impact actions, described in Section VI, defaults to the more conservative of these two postures whenever a region cannot confirm it holds current policy, since acting on a stale governance state is a worse outcome than briefly delaying an automated action during an already-degraded network condition.

D. Cloud-Native Deployment Sketch

A typical deployment runs independent microservices on a container orchestration platform [1], [2]. It uses horizontal autoscaling based on ingest lag and inference latency, service-mesh-enforced mutual TLS [11], GitOps-based delivery, and dedicated node pools that separate streaming, analytical query, and inference workloads. The platform emits its telemetry, covering collector health, processing lag, drop rates, and governance failures, back into itself, achieving observability of the observability layer.

8. Evaluation Methodology and Illustrative Results

A. Framing

This section specifies how an implementation of the architecture should be validated and reports an illustrative simulation constructed for that purpose. The simulation uses a public synthetic identity and event dataset adapted to a telemetry-style workload to establish relative behavior under controlled conditions, borrowing the record-matching and similarity-scoring formulation that deep-learning entity-resolution work has already validated as a reasonable stand-in for correlating structurally similar but non-identical records [23]. It is not a production deployment benchmark, and any implementer adopting this architecture should validate it against their own operational data before drawing production conclusions.

B. Research Questions

Five questions guide the evaluation. Can ingest and correlation sustain target event throughput within latency service-level objectives under burst conditions. Do injected failures preserve durability and bound recovery time. Do schema, lineage, and quality controls catch contract drift and improve investigation reproducibility. Do explainable recommendations improve operator decision time and precision relative to disconnected baselines? Do policy gates prevent disallowed automated actions without an unacceptable loss of legitimate alerts?

C. Workloads and Scenarios

Workload/Scenario	Focus
Steady-state	Baseline latency, throughput, cost
Burst (deploy/incident)	Autoscaling, backlog drain
Failure injection	RTO, data loss, isolation
Multi-region partition	Local continuity, sync lag
Schema evolution	Compatibility, historical readability

Table V. Workload and Scenario Matrix

D. Metrics

Operational metrics cover ingest rate, processing lag, and query latency. Reliability metrics cover availability, replay success, and duplicate handling. Governance metrics cover schema validation rate, lineage coverage, and retention compliance. Operational-intelligence metrics cover precision and recall of detected anomalies, correlation purity, root-cause ranking quality, explanation completeness, and automation precision under policy.

E. Simulated Comparison

To make the comparison concrete rather than purely narrative, we constructed an illustrative simulation to show how governance-aware correlation behaves compared to two baselines under degraded metadata quality, using the same evidence-aggregation and confidence-discounting logic specified for the Operational Intelligence Engine, applied here to a synthetic entity-matching workload as a structural proxy for telemetry correlation under incomplete metadata. Under baseline conditions, a rule-based correlation baseline reached an F1 score of 0.924, a similarity-only automated baseline reached 0.998, and the governance-aware approach reached 0.992. When 35 percent of contextual

metadata fields were randomly withheld to simulate the incomplete tagging that commonly occurs during rapid deployment, the rule-based baseline dropped to an F1 score of 0.605, a decrease of nearly 32 points, while the governance-aware approach remained at 0.966, a decrease of only 2.5 points, and its rate of routing ambiguous cases to human review rose from 3.3 percent to 8.7 percent instead of silently misclassifying them. This pattern is consistent with the architecture's central claim: embedding governance and metadata-quality signals into the confidence model degrades gracefully as input quality drops, where a rigid rule-based approach degrades sharply and without warning.

Model	F1 (Baseline)	F1 (Degraded Metadata)
Rule-based/deterministic baseline	0.924	0.605
Similarity-only automated baseline	0.998	0.996
Governance-aware approach (proposed)	0.992	0.966

Table VI. Simulated Comparison under Baseline and Degraded-Metadata Conditions

F. Threats to Validity

Workload realism, cardinality explosion, label noise around what constitutes a genuine incident, and organization-specific topology all limit how far these results generalize. Explainability and operational-intelligence quality more broadly are difficult to reduce to a single numeric score and likely need human-rated rubrics alongside whatever quantitative metrics are collected.

9. Discussion

A. Comparative Positioning

The architecture should be judged on whether its contracts reduce integration ambiguity across a telemetry estate, not on a feature-by-feature checklist against any named commercial product.

B. Limitations

Implementation choices dominate latency and cost outcomes far more than the reference architecture itself does. Full-fidelity event retention is expensive, so sampling policies are necessary in practice and must themselves be governed rather than left ad hoc. Operational-intelligence quality depends heavily on label accuracy and topology correctness. Governance maturity is fundamentally an organizational property, not solely a technical one, so the architecture can enable but not substitute for stewardship discipline. Not every enterprise needs, or should adopt, full closed-loop automation.

A further limitation concerns adoption cost. Retrofitting schema stamping and lineage capture onto an existing telemetry pipeline that was never designed for governance is a substantial migration effort, and organizations should expect the acquisition and governance layers to take longer to mature than the visualization layer that typically motivates the initial investment. Teams that adopt only the layers visible to end users, while skipping the governance layer that makes those layers trustworthy, will reproduce the same fragmentation that this architecture was designed to eliminate.

C. Future Work

Promising extensions include stronger causal-inference or graph-based root-cause analysis with formal evaluation protocols, federated operational intelligence spanning separate administrative domains, privacy-preserving analytics applied to sensitive log content, large-language-model copilots for incident narratives that mandate citation of underlying telemetry evidence, digital-twin simulation for what-if remediation planning, and community-maintained profiles mapping the seven layers onto concrete open-source reference implementations.

10. Conclusion

This work set out to answer how cloud-native platform engineering, distributed telemetry processing, enterprise metadata governance, and explainable AI-assisted operations can be integrated into a single reference architecture that supports scalable and auditable enterprise observability. The proposed seven-layer architecture answers that question by binding acquisition, streaming, lakehouse storage, governance, operational intelligence, and automation under one set of contracts so that no layer can silently drift out of alignment with the others.

What this architecture contributes beyond existing practice is an explicit refusal to treat governance and explainability as optional hardening layered onto AIOps after deployment. Prior approaches typically advance instrumentation, storage, or anomaly detection in isolation, leaving an operator to reconstruct trust in a recommendation after the fact, often during an active incident when that reconstruction is least convenient. By requiring an explanation artifact for every recommendation and gating automation behind governance prerequisites rather than confidence thresholds alone, this architecture shifts explainability and accountability from an afterthought to a structural default.

Organizations do not need to adopt every layer simultaneously. A staged path, beginning with schema-enforced acquisition and lineage capture, followed by correlated semantic zones, and only then explainable automation behind human approval, allows the architecture to deliver value incrementally while avoiding the common failure of building visible dashboards on top of ungoverned data. As enterprises continue to operate increasingly distributed, multi-region systems, the governance discipline this architecture requires stops being a nice to have and becomes a precondition for operations that can be trusted, audited, and improved over time.

Declarations

Funding: This research received no external funding.

Conflict of Interest: The author declares no conflict of interest.

Data Availability: The synthetic benchmark data used for the illustrative simulation in this manuscript is publicly available through the open-source Python Record Linkage Toolkit. No proprietary or production data was used in this study.

Ethics Approval: Not applicable. This study did not involve human subjects, personal data, or animal research.

Author Contributions: The author was solely responsible for the conceptualization, architecture design, methodology, simulation, writing, and review of this manuscript.

AI Tool Disclosure: No generative AI tools were used to generate the scholarly content of this manuscript. The author affirms responsibility for the entirety of the content.

REFERENCES

1. Brendan Burns, Joe Beda, and Kelsey Hightower, "Kubernetes: Up and Running", O'Reilly, 3rd ed., 2022. Available: https://www.oreilly.com/library/view/kubernetes-up-and/9781098110192/
2. Camille Fournier and Ian Nowland, "Platform Engineering", O'Reilly Media, Inc., 2024. Available: https://www.oreilly.com/library/view/platform-engineering/9781098153632/
3. Brendan Gregg, "Systems Performance", Pearson, 2nd ed., 2020. Available: https://www.brendangregg.com/systems-performance-2nd-edition-book.html
4. Cindy Sridharan, "Distributed Systems Observability", O'Reilly, 2018. Available: https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/
5. OpenTelemetry Authors, "OpenTelemetry Specification", 2024. Available: https://opentelemetry.io/docs/specs/
6. Yingnong Dang, Qingwei Lin, and Peng Huang, "AIOps: Real-world Challenges and Research Innovations", Proc. ICSE-Companion, 2019. Available: https://www.microsoft.com/en-us/research/publication/aiops-real-world-challenges-and-research-innovations/
7. Jasmin Bogatinovski et al., "Artificial Intelligence for IT Operations (AIOps) Workshop White Paper", arXiv:2101.06054, 2021. Available: https://arxiv.org/abs/2101.06054
8. Michael Armbrust et al., "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics", Proc. CIDR, 2021. Available: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

9. Michael Armbrust et al., "Delta Lake: High-performance ACID Table Storage over Cloud Object Stores", Proc. VLDB Endowment, vol. 13, no. 12, pp. 3411 to 3424, 2020. Available: https://doi.org/10.14778/3415478.3415560
10. Benjamin H. Sigelman et al., "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure", Google Technical Report, 2010. Available: https://research.google/pubs/pub36356/
11. William Morgan et al., "Istio Documentation", 2024. Available: https://istio.io/latest/docs/
12. Jay Kreps, Neha Narkhede, and Jun Rao, "Kafka: A Distributed Messaging System for Log Processing", Proc. NetDB, 2011. Available: https://notes.stephenholiday.com/Kafka.pdf
13. Seth Gilbert and Nancy Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", ACM SIGACT News, vol. 33, no. 2, pp. 51 to 59, 2002. Available: https://doi.org/10.1145/564585.564601
14. Martin Fowler, "Event Sourcing and CQRS", martinfowler.com, 2005 to 2011. Available: https://martinfowler.com/eaaDev/EventSourcing.html
15. Sam Newman, "Building Microservices", O'Reilly, 2nd ed., 2021. Available: https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/
16. Prometheus, "Prometheus Documentation", 2024. Available: https://prometheus.io/docs/
17. Cloud Native Computing Foundation, "Cloud Native Definitions and Landscape", 2024. Available: https://www.cncf.io/
18. Nicole Forsgren, Jez Humble, and Gene Kim, "Accelerate", IT Revolution, 2018. Available: https://itrevolution.com/product/accelerate/
19. Niall Richard Murphy, Betsy Beyer, Chris Jones, and Jennifer Petoff, "Site Reliability Engineering", O'Reilly, 2016. Available: https://www.oreilly.com/library/view/site-reliability-engineering/9781491929117/
20. Ann Cavoukian, "Privacy by Design: The 7 Foundational Principles", Information and Privacy Commissioner of Ontario, 2009. Available: https://prod.iapp.org/media/pdf/resource_center/pbd_implement_7found_principles.pdf
21. Roozbeh Aghili, Heng Li, and Foutse Khomh, "Studying the Characteristics of AIOps Projects on GitHub", Empirical Software Engineering, vol. 28, no. 6, art. 143, 2023. Available: https://doi.org/10.1007/s10664-023-10382-z
22. Radoslav Furnadzhiev, Mitko Shopov, and Nikolay Kakanakov, "Efficient Orchestration of Distributed Workloads in Multi-Region Kubernetes Cluster", Computers, vol. 14, no. 4, art. 114, 2025. Available: https://doi.org/10.3390/computers14040114
23. Muhammad Ebraheem et al., "DeepER: Deep Entity Resolution", arXiv:1710.00597, 2018. Available: https://arxiv.org/abs/1710.00597