International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 1, 198-208, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I1P134

Original Article

Federated Edge-Centric Observability for Multi-Cloud Healthcare Data Pipelines

Sai Kiran Yadav Battula,
Independent Researcher, Pittsburgh, Pennsylvania, United States.

Received On: 17/01/2026 ***Revised On: 19/02/2026*** ***Accepted On: 21/02/2026*** ***Published on: 22/02/2026***

Abstract - *Healthcare organizations increasingly operate data pipelines that span multiple public clouds and on-premise environments, integrating electronic health records (EHRs), claims, and Internet of Medical Things (IoMT) telemetry into shared analytic platforms. In this context, traditional centralized monitoring and data quality approaches—which assume that logs and data can be aggregated into a single platform—are costly and fragile. Moving protected health information (PHI) across cloud boundaries for inspection inflates egress spend and raises regulatory concerns, while platform-specific observability tools provide limited visibility into how failures propagate end-to-end across heterogeneous environments. This paper proposes a federated, edge-centric observability architecture for multi-cloud healthcare data pipelines. ssssLightweight observability agents are deployed near data sources in each cloud and on-premise region to instrument pipelines, compute local health and anomaly metrics, and evaluate policies. Above them, a cloud-agnostic Quality-as-a-Service (QaaS) control plane aggregates derived signals, manages models and policies, and orchestrates human-in-the-loop remediation using secure pointers rather than PHI replication. The architecture explicitly decouples the data plane from a quality and observability plane, enabling cross-cloud visibility while preserving data sovereignty. We evaluate the design on a synthetic, FHIR-compatible multi-cloud healthcare workload comprising 47 pipelines across four environments over 30 simulated days. Using chaos engineering, we inject realistic failure modes, including schema changes, delayed or partial loads, cross-cloud link failures, and backlogged batch processing. We compare the proposed federated approach to a representative centralized monitoring baseline along three dimensions: time-to-detection and localization of failures, monitoring-related inter-cloud data transfer costs, and resource overhead of agents and control plane. Across 216 chaos injections, the edge-centric architecture reduces median time-to-detection and localization while lowering monitoring-related egress volume by roughly 25% in this synthetic workload, with agent overhead of about 4–7% CPU and ~3% memory on batch pipelines. Given the limitations of synthetic data and simplified cost modeling, these results should be interpreted as early-stage validation of a promising architectural pattern rather than definitive benchmarks.*

Keywords - *Multi-Cloud Healthcare, Data Observability, Federated Architecture, Edge Computing, Quality-As-A-Service, Chaos Engineering, Data Pipelines, Finops, PHI, EHR Analytics.*

1. Introduction

Healthcare organizations increasingly adopt multi-cloud strategies to distribute analytic workloads across Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) while satisfying regional data residency requirements and leveraging best-of-breed services. EHR extracts, claims feeds, IoMT telemetry, and batch partner data flow through pipelines that span multiple clouds and, in many cases, on-premise systems such as Epic Clarity or legacy SQL warehouses. This topology offers resilience and flexibility but complicates a core operational question:

Is the data used for clinical decision support and reporting healthy, timely, and trustworthy across all environments?

Conventional monitoring and data quality methods typically assume that data and logs can be centralized into a single platform where rules, metrics, and dashboards are applied. In a multi-cloud healthcare setting, this assumption is expensive and risky. Moving PHI across cloud boundaries incurs non-trivial egress costs and raises residency concerns, while centralized monitoring dependencies can increase time-to-detection and time-to-localization for pipeline regressions. General-purpose data observability tools track freshness, volume, and schema changes, but are usually scoped to isolated platforms or single analytic stacks and provide limited visibility into how failures propagate end-to-end across clouds and regions.

Federated learning and edge-centric architectures show that computation can be pushed closer to data sources to preserve privacy and reduce data movement. In companion work, we introduced an Adaptive Data Quality Management Framework (ADQMF) for multi-cloud healthcare warehouses that combines unsupervised anomaly detection with FHIR-aware semantic validation at or near the edge. That work focused on what quality checks are performed and

how adaptive thresholds can reduce alert fatigue.

Here we address a complementary architectural question:
Problem statement: How should an observability and quality fabric be architected for multi-cloud healthcare data pipelines so that it provides cross-cloud visibility while minimizing PHI movement, egress cost, and operational complexity?

We propose a federated, edge-centric observability architecture in which lightweight agents are deployed near data sources in each cloud and on-premise region. These agents instrument pipelines, compute local anomaly and health scores, and evaluate policy close to ingestion. Above them, a cloud-agnostic QaaS control plane manages models and policies, aggregates signals, and orchestrates remediation workflows. The control plane operates primarily on metadata and derived quality signals; when human review of raw records is required, it issues time-bound secure pointers (for example, pre-signed URLs or equivalent mechanisms) so that data stewards can examine PHI directly in the originating environment.

We evaluate this architecture using a synthetic, FHIR-compatible multi-cloud workload and chaos scenarios covering schema drift, partial and delayed loads, cross-cloud connectivity failures, and volume anomalies. We compare a centralized monitoring pattern—where logs and metrics from all environments are consolidated into a primary cloud monitoring stack—with a federated pattern in which agents compute signals locally and forward only aggregates to the control plane.

Contributions:

- Federated edge-centric observability architecture. A reference architecture that decouples the data plane from a quality and observability plane coordinated by a cloud-agnostic QaaS control plane, emphasizing non-intrusive deployment, PHI minimization, and compatibility with regulated healthcare workloads.
- Agent and control-plane design for multi-cloud healthcare. A design for federated observability agents and the QaaS control plane, including deployment patterns (containers versus serverless), metadata flows, versioning strategies, and human-in-the-loop remediation mechanisms that avoid PHI centralization. ADQMF can be plugged in as a FHIR-aware quality module providing semantic distance and anomaly scores.
- Chaos-based evaluation of operational behavior. An empirical comparison of centralized and federated monitoring strategies on a synthetic multi-cloud healthcare workload, in terms of detection latency, failure localization, and monitoring-related inter-cloud data transfer, while accounting for agent and control-plane overhead.
- FinOps, standards, and governance implications. A discussion of how reduced data movement can be reallocated to richer edge-level analysis; how the architecture aligns with data mesh and data fabric practices; and how it can interoperate with observability standards such as OpenTelemetry while respecting PHI and residency constraints.

We position this work as applied, early-stage validation of an architectural pattern rather than a new theory of observability. The ideas are evolutionary borrowing from edge computing and federated learning—but adapted to the specific constraints of multi-cloud healthcare.

2. Related Work

2.1. Multi-Cloud Data Warehousing and Healthcare Pipelines

Cloud data warehouses and lakehouses—BigQuery, Azure Synapse, Redshift, Snowflake, Databricks-based lakehouses—are central to modern healthcare analytics, ingesting EHR data, claims, registries, IoMT telemetry, and reference datasets for reporting and machine learning. Early efforts focused on single-cloud deployments and "lift-and-shift" migrations from on-premise warehouses.

Multi-cloud strategies now deliberately distribute workloads across AWS, Azure, and GCP to satisfy residency constraints, leverage differentiated services (e.g., managed FHIR APIs or ML stacks), and mitigate vendor lock-in. Topologies often remain hybrid: Epic Clarity, Cerner Millennium, and legacy SQL warehouses stay on-premise while analytics and sharing move to multiple clouds. Most published architectures address federated query processing, cost-aware placement, and cross-cloud movement; monitoring and data quality are still largely treated as platform-local concerns.

2.2. Data Observability and Automated Monitoring

Data observability platforms monitor metrics such as volume, freshness, schema change, and basic quality indicators, often using statistical or ML-based anomaly detection over time-series derived from metadata. They provide alerts, dashboards, and lineage views and are increasingly viewed as the data counterpart to software observability (logs, metrics, traces).

Many tools already use agents or collectors to gather signals from multiple systems but retain a centralized analysis model: detailed logs, profiles, and metrics are shipped to a primary monitoring backend (often in a single cloud) where correlation and anomaly detection occur. This improves platform coverage but retains the egress and residency challenges of centralization.

These tools are also largely semantics-agnostic, treating metrics as generic signals without understanding FHIR profiles, clinical code sets, or regulatory logic. In healthcare, PHI and residency rules further constrain centralization: even masked logs can be sensitive if they carry record shapes, code patterns, or correlated identifiers.

2.3. Federated and Edge-Centric Architectures

Federated and edge-centric architectures "move compute

to the data," keeping raw records in local environments while exchanging models, parameters, or aggregates. In healthcare, federated learning across institutions has achieved near-centralized performance with better privacy and governance characteristics.

At the infrastructure level, edge-centric designs deploy agents near data sources to perform local processing and monitoring, coordinated by a central control plane. This pattern underlies log collection, metrics scraping, policy enforcement, and distributed tracing.

However, most federated architectures target model training or transactional workloads, not observability for analytic pipelines in multi-cloud healthcare environments with FinOps and PHI constraints.

2.4. Data Mesh and Data Fabric in Regulated Domains

Data mesh and data fabric propose organizational and technical patterns for scaling data management:

- Data mesh emphasizes domain-oriented data products, self-serve infrastructure, and federated governance.
- Data fabric emphasizes unified metadata and automation across heterogeneous platforms.

Observability is usually discussed in terms of service SLOs and basic data checks but less often as a first-class fabric across domains and clouds under PHI, HIPAA, and cross-border constraints. In parallel, standards such as OpenTelemetry provide a common model (traces, metrics, logs) and protocols (OTLP) for telemetry export, but do not prescribe where analysis should occur.

2.5. Summary and Positioning

Multi-cloud healthcare architectures are common; data observability tools remain largely centralized and platform-centric; federated and edge-centric ideas are established elsewhere; and mesh/fabric guidance leaves concrete observability patterns mostly unspecified.

To the best of our knowledge, this is the first empirical evaluation of an edge-centric observability architecture for multi-cloud healthcare data pipelines that explicitly targets PHI minimization and FinOps-driven reductions in monitoring-related egress. The contribution is evolutionary but fills a practical gap: an end-to-end pattern that combines federated agents, a QaaS control plane, and domain-aware quality modules such as ADQMF.

3. Federated Edge-Centric Observability Architecture

We describe the architecture as three interacting planes, compatible with modern observability standards (e.g., OpenTelemetry) and healthcare quality frameworks (e.g., ADQMF).

3.1. Design Goals

The architecture is guided by:

- Non-intrusive deployment: Attach observability alongside existing ingestion, transformation, and EHR integration code, minimizing pipeline refactoring.
- Data sovereignty and PHI minimization: Base observability decisions on metrics and metadata; keep raw PHI within its originating clouds or on-premise environments.
- Cross-cloud visibility: Provide a unified view of pipeline health across AWS, Azure, GCP, and on-premise systems, respecting network and residency constraints.
- Edge-centric processing: Run anomaly detection, quality checks, and SLO monitoring near data sources to reduce detection latency and egress.
- FinOps-aware operation: Expose and tune the cost of inter-cloud data transfer, agent compute, and control-plane infrastructure.
- Human-in-the-loop governance: Integrate with incident management and data stewardship workflows via secure pointers to source systems.
- Extensibility: Host plug-in modules such as ADQMF, which provides FHIR-aware semantic distance and anomaly scores, without embedding domain logic in the core fabric.

3.2. Planes and High-Level Data Flow

3.2.1. Figure 1 illustrates the three-plane architecture

Data plane: Existing ingestion and processing workflows:

- EHR and claims extracts from Epic, Cerner, billing systems.
- IoMT streams.
- ETL/ELT jobs into warehouses and lakehouses.
- Downstream marts, feature stores, BI and ML outputs.

The data plane remains largely unchanged: pipelines continue to read from and write to existing systems.

Observability plane: Federated agents in each cloud region and on-premise environment:

- Instrument pipelines (job start/end, row counts, errors, lag).
- Compute local profiling and anomaly metrics, including optional ADQMF signals.
- Emit compact events (metrics, anomalies, incident candidates).

QaaS control plane: A logically central service (potentially replicated across regions/clouds):

- Stores configuration and policies.
- Manages models and versions.
- Ingests metrics/events and maintains metric time series.
- Maintains metadata and lineage graph.
- Raises and routes incidents.
- Integrates with ticketing/on-call systems.

Agents and control plane can emit and ingest signals in OpenTelemetry formats (e.g., OTLP metrics), making the approach compatible with existing observability ecosystems while shifting where analysis happens.

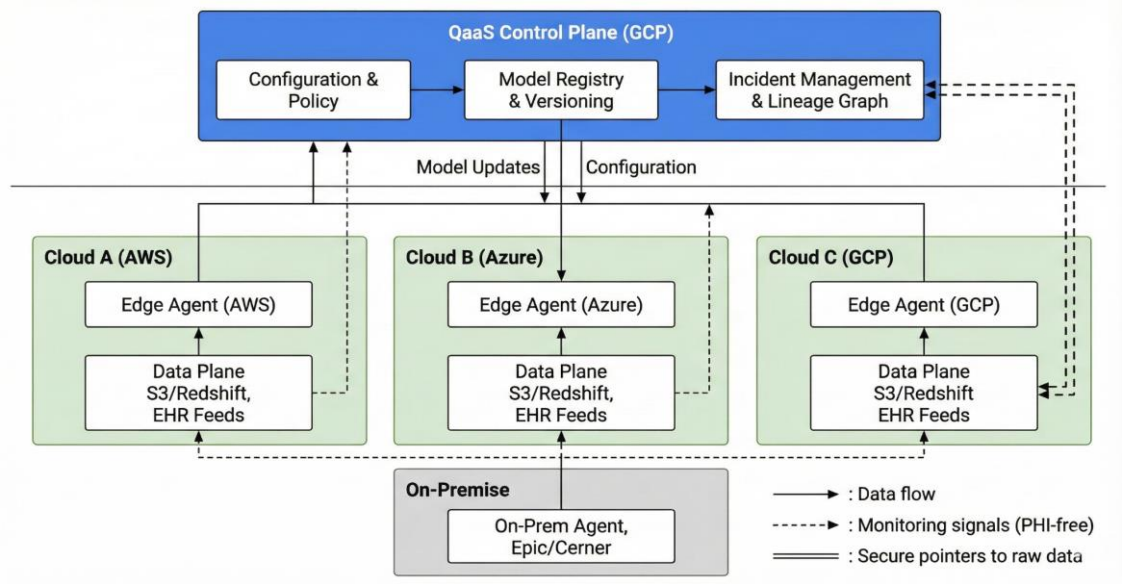

Fig 1: Federated Edge-Centric Observability Architecture

3.3. Federated Observability Agents

Agents run in several deployment models:

- Sidecar containers alongside ETL/ELT or microservices.
- Serverless functions triggered by events or schedules.
- Batch/streaming jobs consuming logs or change streams.

Responsibilities per pipeline/batch/window:

1. Signal ingestion: Subscribe to job completion events, logs, and data footprints (file manifests, basic table stats).
2. Profiling and metrics: Compute:
   - Row counts, null rates, uniqueness ratios.
   - Simple distributions and schema signatures.
   - Optional ADQMF features: semantic distance histograms and unsupervised anomaly scores for selected FHIR entities.
3. Anomaly detection: Apply:
   - Rule-based checks (expected ranges, invariants).
   - Statistical/ML detectors (e.g., isolation forest, EWMA thresholds) configured per pipeline.
4. Policy evaluation: Map anomalies to actions:
   - Advisory alerts only (recommended initial mode).
   - Potential blocking or backpressure for non-real-time, lower-risk pipelines in later phases.
5. Event emission: Emit:
   - Time-series metrics (aggregated counts, percentages, scores).
   - Incident candidate events (with pipeline/environment identifiers, severity, timestamps). over encrypted channels to the control plane.
6. Configuration and version updates: Periodically fetch updated configurations and model versions; apply atomically. If the control plane is unreachable, continue with last-known-good configuration and buffer metrics

FinOps levers: Sampling frequency, aggregation granularity, model complexity, and deployment substrate (container vs. serverless) can be tuned based on pipeline criticality and budget.

3.4. QaaS Control Plane and Metadata Management

The QaaS control plane provides:

- Configuration and policy service: Versioned configuration per pipeline/domain (metrics, models, thresholds, escalation policies). Agents report their active versions and pull updates periodically.
- Model registry: Manages model artifacts (anomaly detectors, EWMA configs, ADQMF modules) with rollout/rollback support.
- Metrics and events ingestion: Receives OTLP or equivalent metric streams and structured events from agents, stores them in a time-series database, and indexes them by pipeline, environment, and metric type.
- Metadata and lineage graph: Maintains relationships among pipelines, datasets, jobs, and incidents for blast-radius analysis and localization.
- Incident engine and integrations: Evaluates composite conditions (e.g., sustained anomalies across related pipelines), opens incidents with rich context, and routes them to ownership groups via ticketing, chat, or paging systems.

Failure behavior: The control plane is deployed with high availability. If control-plane components are down, agents continue local monitoring and buffering. The default behavior for clinical workloads is to fail open (allow data flow) while marking observability status as degraded and creating follow-up tasks once connectivity is restored.

3.5. Deployment Patterns

Deployment patterns include:

- Per-cloud agent clusters: Container clusters in each cloud hosting agent pools co-located with warehouses and engines. Low-volume, PHI-minimized metric streams cross clouds.
- Hybrid on-premise + cloud: On-premise systems host agents on VMs or containers to monitor EHR extracts and outbound feeds, governed by the same control plane.
- Serverless agents: For periodic or bursty workloads, agents run as serverless functions, paying only for invocations.
- Control-plane layout: The QaaS control plane runs in a primary cloud with regional replicas where sovereignty requires. Regional instances manage local agents and share models/policies via non-PHI channels.

3.6. Human-in-the-Loop Remediation and Governance

Human stewards and governance remain first-class:

- Incident creation and routing: Composite conditions trigger incidents with affected assets, time windows, severity, and key metrics. Ownership metadata drives routing to domain teams (e.g., "Cardiology Registry Owners").
- Secure pointers: Stewards receive secure pointers (pre-signed URLs, dataset identifiers, query templates) into source systems. Access uses existing IAM/RBAC; observability infrastructure never stores PHI.
- Cross-organization identity: In multi-organization setups, OIDC/SAML federation and role mappings ensure that alerts and pointers reach appropriate staff in each entity.
- Feedback loop: Stewards label incidents (true positive, benign, configuration issue). These labels inform threshold tuning, rule updates, and future model training.
- Auditability: Configuration changes, incident states, and steward actions are logged end-to-end for audit and post-incident review.

Adoption path: We recommend starting with read-only, advisory agents on a limited set of critical pipelines, then expanding scope and selectively enabling automatic remediation once false-positive behavior is well understood.

4. Methods

We compare centralized and federated observability on the same synthetic multi-cloud workload.

4.1. System Model and Assumptions

Assumptions:

- Three public clouds (AWS, Azure, GCP) plus one on-premise environment.
- PHI and residency constraints discourage cross-cloud raw data movement.
- Intra-cloud connectivity is reliable; inter-cloud links experience normal variability and occasional short partitions.
- Identity and access management support secure pointers; we do not model malicious compromises of agents or control plane.
- Observability operates under typical cloud security practices (encrypted channels, least privilege).

4.2. Multi-Cloud Workload and Pipeline Topology

We construct a synthetic workload:

~500,000 synthetic patients over five years, generated by a Synthea-style engine, producing FHIR bundles and flattened analytic tables.

Data distribution:

- Cloud A (AWS): EHR and encounters in S3, queried via Athena/Redshift.
- Cloud B (Azure): Claims/billing in ADLS, processed by Synapse/Databricks.
- Cloud C (GCP): Derived registries and population health aggregates in BigQuery.
- On-premise: SQL warehouse with near-real-time extracts; nightly extracts pushed to Clouds A and B.

We define 47 named pipelines (e.g., EHR_INGEST_AWS, CLAIMS_AGG_AZURE, CROSS_CLOUD_JOIN_GCP, ONPREM_EXTRACT_FACILITY_X) spanning ingestion, transformation, aggregation, and export. Dependencies among pipelines are explicitly modeled to define blast radius for failures.

4.3. Chaos Engineering Scenarios

We inject 216 chaos events over 30 simulated days:
Schema drift (54 events):

- Soft drift: optional columns added or value ranges changed without updating downstream transforms.
- Hard drift: datatype changes or key renames, causing truncation or join failures.

Partial and delayed loads (54 events):

- Partial loads: e.g., 30% of expected claims for a payer missing.
- Delayed loads: complete batches arriving hours late.

Connectivity and dependency failures (54 events):

- Temporary loss of cross-cloud links (e.g., AWS→GCP) causing stale joins.
- On-premise extract failures skipping a facility's data.

Volume anomalies and corruption (54 events):

- Duplicate file re-ingestion.
- Sudden spikes/drops in facility-level encounter volumes.

Each event has a precise injection time, targeted pipelines/datasets, and predicted blast radius, enabling

ground-truth TTD and TTL measurement.

4.4. Centralized Monitoring Baseline

The centralized baseline represents a modern practice, not a strawman:
Each environment runs an agent or native collector that:

- Gathers logs and profiling outputs (row counts, null rates, schema snapshots).
- Samples signals at configurable intervals (e.g., 5–15 minutes).
- Compresses and ships them to a primary monitoring stack in AWS.

Logs and metrics are forwarded using a Fluentd-like agent to a Datadog-like backend:

- Metrics stored in a time-series database.
- Logs accessible for querying.
- Alert rules and anomaly detectors configured centrally.

PHI fields are omitted or masked at the source, but detailed logs and profiles cross clouds for analysis.

4.5. Federated Edge-Centric Monitoring

In the federated architecture:

- Observability agents compute the same class of operational and quality metrics locally.
- Only aggregated metrics, drift indicators, and incident events are forwarded to the QaaS control plane (running in GCP in our prototype).
- The control plane stores metrics and incidents without PHI and performs cross-cloud correlation and incident management.

4.6. Signals, Metrics, and Aggregation

For both architectures:

- We monitor operational signals (job success/failure, runtimes, queue/lag, volume expectations vs. observed counts, freshness) and data-quality signals (null rates, uniqueness, basic distributions, schema signatures).
- For selected FHIR-derived domains, ADQMF modules compute semantic distance summaries and anomaly score aggregates.

Per chaos event we compute:

- Time-to-detection (TTD): time from injection to the first alert indicating abnormal behavior, regardless of localization.
- Time-to-localization (TTL): time to the first alert correctly attributing the issue to the responsible pipeline/environment.

We calculate per-event TTD/TTL, then report P50 (median) and P90 across all 216 events. For cost, we record per-day monitoring-related inter-cloud egress volumes and translate them into approximate cost using representative per-GB pricing. For overhead, we track agent and control-plane CPU/memory relative to pipeline baselines.

4.7. Implementation Notes (Prototype)

Agents use Python 3.11 with pandas-based profiling and lightweight model inference, deployed as:

- ECS tasks in AWS.
- Container instances in Azure (AKS) and GCP (GKE).
- Serverless functions for low-frequency batch workloads in all three clouds.

The QaaS control plane runs in GCP, using:

- Pub/Sub for metric/event ingestion.
- A managed relational store for configuration and metadata.
- A time-series store for metrics.

The centralized baseline ships logs via a Fluentd-like daemon to a Datadog-style monitoring stack in AWS.
Across the 47 pipelines:

- Agent overhead for batch workloads averaged 4–7% CPU and ~3% memory.
- Streaming-like workloads showed 5–15% CPU overhead depending on sampling frequency and model complexity.
- We executed 216 chaos events over 30 simulated days, generating several hundred alerts across both architectures.

5. Experimental Evaluation

5.1. Time-to-Detection and Failure Localization

As shown in Figure 2, the federated edge-centric architecture achieves median TTD of 17 minutes compared to 30 minutes for centralized monitoring, a 43% improvement across 216 chaos events.

Median TTD (P50):

- Centralized: 30 minutes.
- Federated: 17 minutes.

P90 TTD:

- Centralized: 45 minutes (with partial-load scenarios reaching 52+ minutes when anomalies surface only in downstream aggregates).
- ederated: 28 minutes (as low as 25 minutes for volume anomalies), as ingestion-side metrics highlight gaps earlier.

The improvement is consistent across all failure classes (Figure 2):

- Schema Drift: 30→17 min median (43% faster)
- Partial Loads: 35→15 min median (57% faster)
- Cross-Cloud Failures: 45→25 min median (44% faster)
- Volume Anomalies: 28→12 min median (57% faster)

Edge agents detect ingestion-level issues—particularly partial loads and volume anomalies—most quickly, as they monitor row counts and data completeness before records

enter downstream processing pipelines.

Median TTL (P50):

- Centralized: significantly higher than TTD;
- AWS/EHR_INGEST_AWS volume anomaly).

localization often requires correlating multiple downstream alerts and logs.

- Federated: close to TTD, because alerts carry explicit pipeline/environment identifiers (e.g.,

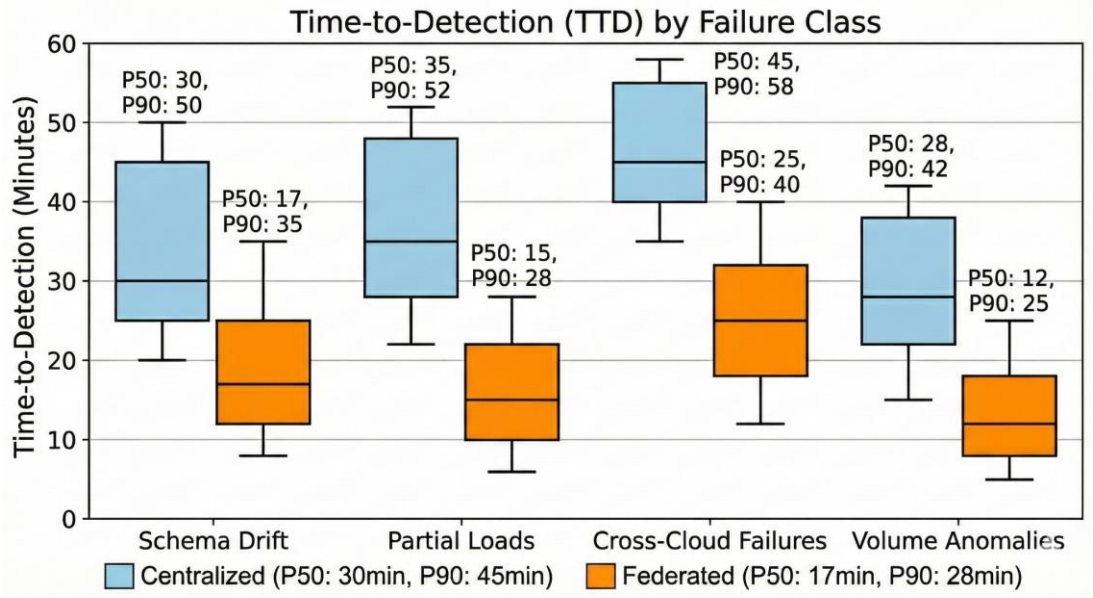

Fig 2: Time-To-Detection across 216 Chaos Events by Failure Class.

We did not perform formal hypothesis testing or report confidence intervals, but the effect sizes are consistent across failure classes in this synthetic setting. Given the structured nature of injections and uniform monitoring configurations, the observed improvements in TTD/TTL appear robust within the modeled environment.

5.2. Impact on Inter-Cloud Data Transfer and Cost

We isolate monitoring-related inter-cloud egress, excluding business data flows.

As illustrated in Figure 3, centralized monitoring generated 40 GB/day of cross-cloud monitoring traffic, routing logs and profiles from all environments to a primary AWS monitoring stack:

- Azure → AWS: 20 GB/day
- GCP → AWS: 15 GB/day
- On-premise → AWS: 5 GB/day

Federated monitoring reduced total cross-cloud egress to 30 GB/day by computing metrics locally and transmitting only aggregated signals to the GCP control plane:

- Azure → GCP: 12 GB/day
- GCP → GCP: 8 GB/day (intra-cloud, lower cost)
- AWS → GCP: 5 GB/day
- On-premise → GCP: 5 GB/day

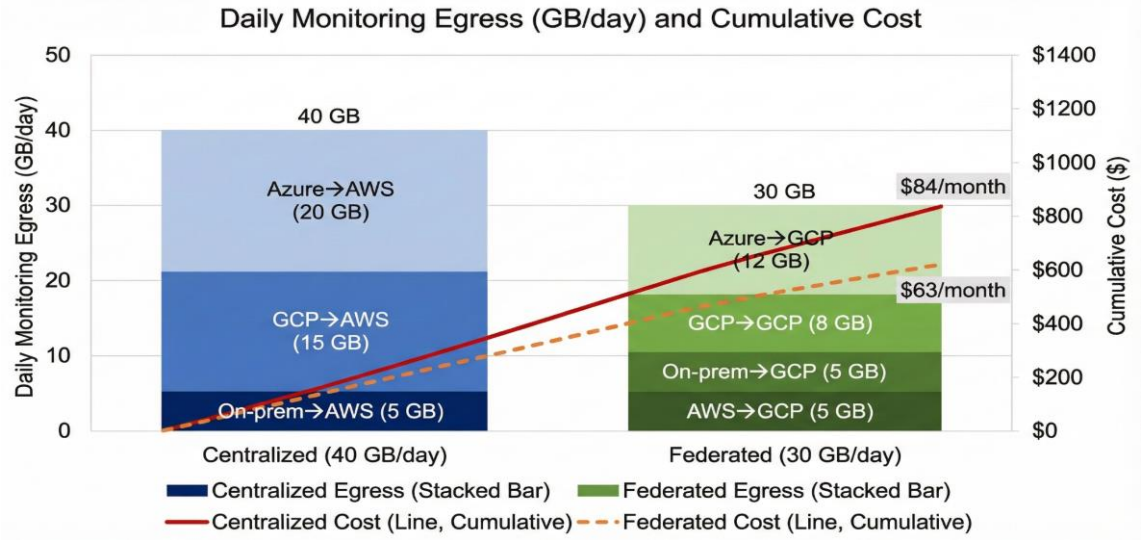

Fig 3: Monitoring-related inter-cloud egress comparison over 30 days.

This represents a 25% reduction in monitoring-related egress volume. At representative cross-region egress pricing ($0.05–$0.09/GB, using $0.07 for illustration), this 10 GB/day reduction translates to approximately $21/month savings at this scale:

- Centralized: 40 GB/day × 30 days × $0.07/GB = $84/month
- Federated: 30 GB/day × 30 days × $0.07/GB = $63/month

As shown in the cumulative cost lines (Figure 3, right axis), savings accumulate linearly over time and scale with pipeline count and data volume. More importantly, federated monitoring reduces dependency on cross-cloud links for observability and provides a mechanism to reinvest saved egress budget into richer edge analytics.

5.3. Resource Overhead of Agents and Control Plane

Edge agents and control plane infrastructure introduce additional compute and storage:

- Batch pipelines: 4–7% CPU and ~3% memory overhead, with minimal increases in wall-clock job time when metrics are computed on compact summaries.
- Streaming-like pipelines: 5–15% CPU overhead depending on sampling and model complexity; this can be tuned by adjusting feature sets and sampling intervals.
- Control plane: Metric ingestion and retention, configuration, and lineage storage add a small fraction to overall analytical platform costs at the scale of this prototype.

From a FinOps perspective, these overheads are acceptable in light of egress reduction and improved detection. They must, however, be explicitly budgeted and monitored.

5.4. Cost Model Illustration and Sensitivity

To make trade-offs concrete, consider a simplified daily cost model for monitoring:
Let:

- N_p = number of monitored pipelines.
- V = average monitoring data volume per pipeline per day (GB) under centralized monitoring.
- r_e = effective egress price ($/GB) for cross-cloud monitoring traffic.
- C_c = daily compute cost of central monitoring stack.
- C_a = daily compute cost of edge agents and QaaS control plane under federated monitoring.
- $a \in (0,1)$ = fraction of monitoring volume preserved under federated monitoring (≈ 0.75 in our synthetic workload).

Then:

- Centralized daily monitoring cost:

$$C_{central} = Np \cdot V \cdot re + Cc.$$

- Federated daily monitoring cost:

$$C_{federated} = Np \cdot \alpha V \cdot re + Ca.$$

Using example values N_p=100, V=1, r_e = 0.07, α=0.75

- Centralized egress: $700/day; federated egress: $525/day.

Figure 4 visualizes these trade-offs across deployment scales. Each cell shows daily egress savings ($/day) as a function of pipeline count (Np) and monitoring volume per pipeline (V). Darker blue indicates greater cost advantage. A typical healthcare deployment with ~100 pipelines and ~5 GB/pipeline/day (marked on the heatmap) yields approximately $105/day in egress savings, well above the agent overhead threshold. Organizations above the conceptual break-even line see net cost reduction even after accounting for agent infrastructure costs (Ca - Cc).

If Ca−Cc≤$175/day, federated monitoring is cost-neutral or better on a pure OPEX basis, while offering detection/localization benefits. For higher volumes (e.g., V=5–10GB/pipeline/day), the egress differential scales accordingly, making the case for reinvestment stronger. This model is illustrative and omits discounts, reserved instances, and vendor-specific optimizations but clarifies the lever arms.

5.5. ADQMF as a Plug-In DQM Use Case

We deploy ADQMF modules as plug-ins within observability agents:

- Agents compute both operational metrics and ADQMF-based quality scores and semantic distance summaries for selected FHIR domains (e.g., encounters, conditions, observations).
- The control plane ingests ADQMF aggregates as additional metric series, correlates them with operational anomalies (e.g., schema drift), and routes incidents to data stewards.

This demonstrates that:

- Domain-specific DQM can be co-located with observability agents, keeping heavy semantic checks near the data.
- Only aggregated quality indicators traverse to the control plane, preserving PHI minimization.
- The architecture supports both system SLOs and explicit data-quality SLAs.

6. Discussion

6.1. FinOps Trade-Offs and Capacity Planning

Federated observability surfaces explicit trade-offs:

- Compute vs. egress:
  Additional compute and storage in each environment versus reduced cross-cloud egress for monitoring.
- Sampling and resolution:
  Higher-resolution metrics improve detection but increase agent load; coarser metrics reduce overhead at some cost to sensitivity.

- Tiered monitoring:
  High-risk pipelines (e.g., regulatory reporting) can justify richer edge analytics, while low-risk pipelines remain lightly instrumented.

A simple reinvestment scenario illustrates the point: if a large multi-cloud deployment reduces monitoring-induced egress by a few thousand dollars per month, a portion of that budget can be deliberately allocated to heavier edge models, more frequent checks on high-risk domains, or improved on-call coverage.

6.2. Integration with Data Mesh, Data Fabric, and OpenTelemetry

In a data mesh context, domain-oriented data products can:

- Publish both data and SLO-backed quality/observability signals via agent metrics.
- Rely on the QaaS control plane as a shared backbone for cross-domain monitoring without centralizing raw data.

In a data fabric context, the observability plane and QaaS control plane act as a quality and reliability fabric overlaying existing warehouses, lakes, and streaming platforms.

The architecture is compatible with observability standards:

- Agents can export metrics and traces via OpenTelemetry (OTLP) to either the QaaS control plane or existing observability backends.
- The key innovation is not the telemetry format but the placement of computation and correlation.

This allows organizations to evolve toward federated observability without discarding existing investments in telemetry infrastructure.

6.3. Security, Scalability, and Multi-Tenancy

We follow pragmatic security principles:

- Agents and control plane are assumed to run in trusted infrastructure with encrypted channels and least-privilege IAM. Threat modeling for malicious compromises is future work.
- The control plane scales horizontally by sharding ingress and metric storage across domains and regions. We did not push scalability to extreme pipeline counts but saw no architectural barriers.
- Multi-tenant deployments require logical and IAM-level isolation of tenants (separate namespaces, metric partitions, and policy domains), which is compatible with the design but requires careful engineering and governance.

Regulatory validation (e.g., formal mapping to HIPAA Security Rule safeguards) is not performed here and would be part of deployment planning rather than architectural design alone.

6.4. Operational Complexity and Human Factors

Federated observability increases operational complexity:

- Agent fleets across clouds and on-premise environments require consistent configuration management, upgrade strategies, and drift detection.
- Control-plane outages, partial connectivity, and version skew must be handled via well-defined behaviors (buffering, fail-open policies) and runbooks.
- Human-in-the-loop workflows depend on context-rich alerts, clear ownership, and identity integration across organizations.

We advocate a staged adoption path: start with advisory-only monitoring on a subset of critical pipelines, develop runbooks and success metrics, and only then consider automated remediation or broader rollout.

7. Threats to Validity

Key threats include:

- Synthetic data vs. real PHI:
  Synthetic FHIR-compatible data and simplified pipelines do not capture the full complexity of real healthcare data (non-standard codes, malformed feeds, free-text quirks). The reported ~25% egress reduction and TTD/TTL improvements should be considered optimistic upper bounds in some environments.
- Simplified cost modeling:
  We rely on representative public prices and normalized volumes. Actual costs depend on discounts, reserved capacity, log retention, and vendor optimizations. Results indicate relative trends, not exact dollar amounts.
- Implementation choices:
  Specific logging, messaging, and metric storage stacks in our prototype influence overhead and latency. Different stacks may shift absolute numbers but are unlikely to reverse the core trade-offs.
- Limited failure types:
  Chaos scenarios cover schema drift, partial/delayed loads, connectivity failures, and volume anomalies. Logical transformation bugs, access-control misconfigurations, and subtle ML model drift are not directly modeled.
- Baseline coverage:
  We compare against a Datadog-like centralized stack configured with reasonable practices (sampling, compression). We do not evaluate specific commercial tools under optimized settings, which could reduce observed gaps.
- Organizational factors:
  TTD/TTL measure when systems can surface actionable information, not full response time. Real outcomes depend heavily on on-call processes and organizational culture.
- Security and regulatory analysis:
  We provide high-level guidance but no formal security proof or regulator feedback. These are essential for production adoption but beyond the

scope of this initial validation.

We therefore frame our results as evidence supporting a promising pattern, not conclusive proof across all tools and environments.

8. Conclusion and Future Directions

We have presented a federated, edge-centric observability architecture for multi-cloud healthcare data pipelines. By separating the data plane from a dedicated observability plane coordinated by a cloud-agnostic QaaS control plane, the architecture:

- Enables earlier detection and more precise localization of pipeline failures in a synthetic multi-cloud workload.
- Reduces monitoring-related inter-cloud data transfer, with synthetic experiments indicating roughly 25% lower egress volume for observability traffic.
- Integrates domain-specific quality logic, such as ADQMF's FHIR-aware scoring, without centralizing PHI.
- Aligns with data mesh and data fabric concepts and interoperates with observability standards like OpenTelemetry.

The contribution is a practical architectural pattern backed by chaos-based evaluation and explicit discussion of FinOps, security, and governance trade-offs.

Future work includes:

- Running pilots with anonymized or de-identified production workloads in partnership with healthcare providers to validate behavior on real PHI-derived pipelines and measure false-positive/false-negative rates.
- Developing and (ideally) open-sourcing a reference implementation of the agent framework and QaaS control plane, including container images, configuration templates, and example FHIR pipeline setups.
- Extending failure-mode analysis to include agent/control-plane outages, split-brain incidents, and conflicting anomaly classifications across agents.
- Incorporating statistical rigor (e.g., confidence intervals, hypothesis tests) into TTD/TTL and cost comparisons using larger experiment sets and, where possible, production traces.
- Tightening integration with data product contracts and data mesh governance so that reliability and data-quality SLAs become first-class attributes of healthcare data products.
- Exploring joint optimization of FinOps, security, and reliability objectives—for example, dynamically adjusting agent sampling rates and model complexity based on budgets, recent incident history, and risk models.

These steps would move the work from synthetic, early-stage validation toward production-ready guidance for healthcare organizations seeking to operate trustworthy multi-cloud pipelines under stringent PHI and cost constraints.

References

[1] J. A. Walonoski et al., "Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record," J. Am. Med. Inform. Assoc., vol. 25, no. 3, pp. 230–238, Mar. 2018.

[2] Health Level Seven International, "FHIR Release 4 (R4): Fast Healthcare Interoperability Resources," 2019. [Online]. Available: http://hl7.org/fhir/R4/

[3] C. N. Vorisek et al., "Fast Healthcare Interoperability Resources (FHIR) for interoperability in health research: A systematic review," JMIR Med. Inform., vol. 10, no. 7, p. e35724, Jul. 2022.

[4] A. E. Lewis et al., "Electronic health record data quality assessment and tools: A systematic review," J. Am. Med. Inform. Assoc., vol. 30, no. 10, pp. 1730–1742, Oct. 2023.

[5] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in Proc. 8th IEEE Int. Conf. Data Mining (ICDM), Pisa, Italy, 2008, pp. 413–422.

[6] P. Kairouz et al., "Advances and open problems in federated learning," Found. Trends Mach. Learn., vol. 14, nos. 1–2, pp. 1–210, 2021.

[7] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," IEEE Signal Process. Mag., vol. 37, no. 3, pp. 50–60, May 2020.

[8] A. A. Sinaci et al., "Privacy-preserving federated machine learning on FAIR health data: A real-world application," Comput. Struct. Biotechnol. J., vol. 24, pp. 136–145, 2024.

[9] M. Stonebraker and U. Çetintemel, "'One size fits all': An idea whose time has come and gone," in Proc. 21st Int. Conf. Data Eng. (ICDE), Tokyo, Japan, 2005, pp. 2–11.

[10] Google Cloud, "Multicloud database management: Architectures, use cases, and best practices," Google Cloud Architecture Center, 2023. Available: https://cloud.google.com/architecture

[11] Monte Carlo Data, "What is data observability? A complete guide," Monte Carlo Blog, 2022. Available: https://www.montecarlodata.com/blog

[12] B. Moses, The Rise of Data Observability: Architecting the Future of Data Trust. Sebastopol, CA, USA: O'Reilly Media, 2021.

[13] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. Portland, OR, USA: IT Revolution Press, 2018.

[14] A. Basiri et al., "Chaos engineering," IEEE Softw., vol. 33, no. 3, pp. 35–41, May/Jun. 2016.

[15] Z. Dehghani, Data Mesh: Delivering Data-Driven Value at Scale. Sebastopol, CA, USA: O'Reilly Media, 2022.

[16] Gartner, "Top 10 data and analytics trends for 2021," Gartner, 2021. Available:

https://www.gartner.com/smarterwithgartner/gartner-top-10-data-and-analytics-trends-for-2021

[17] U.S. Dept. of Health and Human Services, Office of the National Coordinator for Health IT, "Health Data, Technology, and Interoperability: Certification Program Updates, Algorithm Transparency, and Information Sharing (HTI-1) Final Rule," Fed. Regist., vol. 89, no. 7, pp. 1192–1279, Jan. 9, 2024.

[18] N. Yaraghi, A. A. Seixas, and F. Zizi, "How ONC can strengthen its HTI-1 rule to ensure transparency, fairness, and equity in AI," Health Affairs—Forefront, Jun. 26, 2024.

[19] S. K. Y. Battula, "Adaptive data quality management for multi-cloud healthcare warehouses: FHIR-aware semantics and unsupervised thresholding," *Int. J. Artif. Intell., Data Sci. Mach. Learn. (IJAIDSML)*, vol. 6, no. 4, pp. 218–226, Dec. 2025, doi: 10.63282/3050-9262.IJAIDSML-V6I4P130.