Data Platform Architecture for Healthcare Systems

Building a regulated, integrated data foundation that actually holds up

Faris Fyzee | M.Sc. Computer Science, Georgia Tech (AI Specialization)

Executive summary. Healthcare data is fragmented by design. EHR, payer, claims, RCM, clinical, and operational systems each evolved to solve a local problem, and integration was never the priority. A modern data platform in healthcare must reconcile that fragmentation while meeting HIPAA obligations, supporting analytics and AI, and remaining auditable for years after any given dataset is loaded. This paper outlines the architecture patterns that work in practice, the controls that are non-negotiable in a regulated environment, and the organizational habits that separate platforms that hold up from ones that become liabilities.

The fragmentation problem

A typical health system or payer runs data across dozens of systems that were never designed to speak to each other. The fragmentation is not an accident; it is the result of decades of vendor-driven system selection, mergers and acquisitions, and regulatory requirements that produced purpose-built applications. Any platform strategy must start by acknowledging the actual landscape:

- **EHR** (Epic, Cerner/Oracle Health, Meditech, Athena) — deep clinical data, vendor-controlled schemas, limited real-time export, and API access that is improving but still constrained. Large organizations frequently run more than one EHR across acquired entities, each with its own extension fields and customizations.
- **Payer systems** — eligibility, benefits, authorization, adjudication, remit — each often on different refresh cadences, with inconsistent identifiers across systems, and with vendor-specific extensions that do not cleanly map to standard data models.
- **Claims** — 837 submissions, 835 remits, clearinghouse feeds, and payer-specific formats that require normalization before use. The same claim can exist in four different states across four different systems at the same time, and reconciling those states is a recurring engineering problem.
- **RCM and billing** — patient accounting, charge capture, coding, denials management, and collections, frequently on separate platforms with their own reference data, their own user identities, and their own audit trails.
- **Operational systems** — scheduling, staffing, supply chain, referrals, and patient access — which drive cost and throughput but rarely get integrated with clinical data, because the teams that own them report into a different part of the organization.

- **External data** — HIE feeds, SDOH data, pharmacy data, lab reference data, and increasingly, data from remote monitoring and consumer health devices. This data is growing in volume and clinical relevance but rarely has a clean home in existing architectures.

The result: the same patient, provider, or encounter can be represented five different ways across five systems, with no authoritative record. Reconciling those representations — not just technically joining them, but agreeing on which one is canonical — is the foundational work any platform strategy must address. Skipping this step produces a platform that can load data but cannot be trusted to answer a question.

Architecture patterns that work

Three patterns, used in combination, deliver most of the value in regulated healthcare environments. None of them is novel; what matters is executing them with the discipline that regulated data requires.

1. Lakehouse as the core

A lakehouse architecture (Databricks, Snowflake with Iceberg, or an equivalent open-table-format stack) is the practical default. It supports the raw, semi-structured, and structured data healthcare produces — HL7 messages, FHIR bundles, claims EDI, scanned documents, imaging metadata, free-text notes — without forcing premature schema decisions, while preserving ACID guarantees and fine-grained access control. The choice between specific vendors matters less than the architectural principles: open formats, separation of storage and compute, and first-class metadata.

A medallion structure — bronze (raw, immutable), silver (conformed, deduplicated, joined on canonical keys), gold (business-ready, semantically modeled) — works well when enforced strictly. The common failure is letting silver and gold drift into a pile of ad-hoc marts with no lineage, owned by whichever team built them last. Governance over promotion from one layer to the next, with clear ownership and review standards, is what keeps the platform trustworthy over time. Without that discipline, the platform becomes the same fragmented mess it was supposed to replace, just in a new system.

2. Pipelines built for healthcare cadences

Healthcare data arrives on wildly different cadences: EHR events can be near-real-time, claims arrive in daily or weekly batches, remits can lag weeks or months, and reference data (code sets, payer contracts, fee schedules) changes on irregular schedules that have to be managed explicitly. Pipelines must be designed for this reality rather than forcing everything onto a single cadence:

- **Event-driven for clinical data** (HL7 v2, FHIR subscriptions, Kafka or equivalent event streaming) where latency matters for care decisions, workflow integration, or real-time alerting. The cost of this architecture is real; use it where it earns its keep.
- **Batch with strong reconciliation for claims and financial data,** where correctness matters more than speed and late-arriving data is the norm. Reconciliation logic must handle corrections, resubmissions, and adjustments — none of which are edge cases in healthcare finance.

- **Change data capture** from operational systems where direct API access is limited or where the source system cannot be modified. CDC requires careful handling of schema drift and deletes, both of which are common in older healthcare applications.
- **Idempotent, replayable jobs** — every pipeline should be re-runnable from any point without duplication or data loss. This is a hard requirement in an audited environment: when a bug is found six months later, the team must be able to reproduce and correct the data without manual cleanup.
- **Data contracts between producers and the platform.** When a source system changes a field, the platform team needs to know before the change ships. Informal relationships break under pressure; explicit contracts with owners and change-notification SLAs do not.

3. Access patterns and the semantic layer

Raw data is useless to most consumers. A semantic layer — whether through dbt models, a dedicated metrics layer, or a curated FHIR-aligned data model — is what makes the platform usable by analysts, clinicians, quality teams, and downstream AI. Without it, every team rebuilds the same definitions of "encounter," "admission," "inpatient stay," or "denied claim" in slightly different ways, and reporting becomes unreliable — two dashboards disagree, and no one can explain why.

The semantic layer also enforces business logic centrally: risk-adjusted benchmarks, attribution rules, payer-specific definitions. This is where clinical and financial leaders should be directly involved. It is not a data engineering artifact; it is the organization's agreed definition of how it measures itself.

Security, HIPAA, and controls

In a regulated environment, controls are not an add-on. They shape the architecture from day one, and retrofitting them after the fact costs an order of magnitude more than building them in correctly. The non-negotiables:

- **PHI classification and tagging** at the column level, enforced in the platform rather than by convention. Masking, row-level security, and access policies applied automatically based on tags, not based on manual review. Tag propagation through transformations is essential — a derived column from a PHI source is still PHI.
- **Identity and access** tied to enterprise SSO with role-based access, just-in-time elevation for break-glass scenarios, periodic access reviews, and full audit logging. Service accounts should be treated with the same rigor as human identities, not as a convenient way to bypass controls.
- **Encryption at rest and in transit** as table stakes; customer-managed keys where the organization's risk posture requires it. Key rotation procedures should be tested, not just documented.
- **De-identification pipelines** for research, analytics sandboxes, and vendor data sharing — designed so re-identification risk is measurable, not assumed low. Safe Harbor and Expert Determination each have their place; the choice should be documented with the reasoning.
- **Business Associate Agreements** with every vendor that touches PHI, including cloud providers, analytics tools, model vendors, and contractors. The platform should make it easy to enumerate

which systems and which data flows are covered under which BAA, and to detect when a new integration is created outside that inventory.

- **Audit logging** designed for investigators, not just operators. Every access to PHI should be reconstructable six or twelve months later with enough context to answer a regulator's or a patient's question about who saw what and why. Logs that are technically present but impossible to query are not audit logs; they are liability.
- **Environment separation** that is structural, not procedural. Production PHI must not be copyable into development or sandbox environments, regardless of how inconvenient that makes testing. Synthetic and de-identified test data are the answer; shortcuts here create the breaches that end up in the news.

What actually works in regulated environments

A few patterns separate platforms that hold up from ones that become liabilities, and these patterns are organizational at least as much as they are technical:

- **Treat the platform as a product,** with a named product owner, a published roadmap, SLAs to consumers, and a support model. Platforms run as infrastructure projects tend to ossify because there is no one responsible for whether anyone can actually use them.
- **Invest in master data management early** — patient, provider, payer, encounter, and facility identities need authoritative sources and explicit matching rules. This is unglamorous, expensive, and essential. Every significant reporting or AI failure downstream can usually be traced back to an identity resolution problem upstream.
- **Make lineage visible.** Every dataset should answer, without a human in the loop: where did this come from, how was it transformed, who owns it, when was it last refreshed, what is its classification? If this information is not trivially available, the platform is not ready for regulated workloads.
- **Separate environments rigorously.** Production PHI must not leak into dev or sandbox environments, regardless of how inconvenient this makes testing. The answer is investment in synthetic and de-identified test data, not shortcuts.
- **Automate compliance evidence.** Access reviews, audit reports, data flow documentation, BAA inventories, and control attestations should be generated from the platform as a matter of course, not assembled manually in the weeks before each audit. Manual assembly is where errors and omissions happen.
- **Resist the urge to consolidate everything immediately.** A pragmatic platform connects high-value domains first — claims + RCM to drive revenue cycle work, or EHR + scheduling to drive operational efficiency — and expands from there. Attempts to unify everything at once fail predictably, because the political and technical costs rise faster than the business value until the program loses support.
- **Plan for cost governance from the start.** Lakehouse and cloud data platforms are operationally expensive. Without attribution of cost to consumers, usage grows unchecked and the finance

team eventually forces a reactive cleanup that damages trust. Build cost reporting in early, even if no one is asking for it yet.

Bottom line

A healthcare data platform is not a technology choice; it is a governance and integration discipline wrapped around a technology stack. The organizations that get it right treat the platform as a long-lived product, embed compliance into the architecture rather than around it, invest in the unglamorous work of master data and lineage, and resist the temptation to solve every integration at once.

The test is simple: can a new analyst, a new clinical program, or a new AI model get the data it needs in days rather than months, under the same controls as the rest of the business, with answers they can defend? If yes, the platform is working. If no, the problem is not the stack — it is the discipline around it.