International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

DataGuard: A Self-Hosted, AI-Accelerated Architecture for Multi-Source Regulatory Data Quality and Governance

Neha Tyagi

Independent Researcher & Platform Architect — Data Analytics, AI, and Regulatory Compliance-New York, USA

Publication History: Received: 04.05.2026; Revised: 03.06.2026; Accepted:08.06. 2026; Published: 19.06.2026

ABSTRACT: Regulated industries — banking, insurance, capital markets, life sciences, and supply chain — are subject to mandatory data submissions governed by frameworks such as the Basel Committee's BCBS 239 principles, the Federal Reserve's Comprehensive Capital Analysis and Review (CCAR), FinCEN anti-money-laundering (AML) obligations, HIPAA, and FDA 21 CFR Part 11. Existing tooling for validating data against these obligations falls into four inadequate categories: manual spreadsheet-based checks that do not scale, commercial data-quality suites that require multi-month implementations and dedicated engineering teams, brittle in-house scripts tied to individual engineers, and vendor-hosted SaaS platforms that raise data-residency concerns for regulated data. This paper presents the architecture of DataGuard, a self-hosted, AI-accelerated data quality and governance platform that integrates five technology domains — large language model (LLM) assisted rule authoring, an eight-database enterprise connectivity layer, cross-source column-level data lineage analysis, a library of seven regulatory rule packs spanning five industry sectors, and cloud-native, zero-infrastructure deployment — into a single deployable application. We describe the platform's layered-monolith architecture, its REST API surface, its four distinct Claude-API-based AI subsystems (including an automated metadata cataloging engine, DocuSmart), its three-tier column-matching algorithm for lineage inference, and its security and governance design, including an OAuth-based Git integration that establishes a version-controlled chain of custody for validation rules and results. We further situate the platform against alternative approaches, walk through an end-to-end validation session on the live application, and discuss the architecture's limitations and directions for future work. The platform is publicly accessible and its source is version-controlled, allowing independent verification of the claims described herein.

KEYWORDS: data quality, data governance, regulatory compliance, data lineage, large language models, BCBS 239, CCAR, enterprise software architecture, metadata management

I. INTRODUCTION

A single data quality failure in a regulatory submission can trigger penalties ranging from hundreds of thousands to hundreds of millions of dollars, mandatory resubmission, heightened supervisory scrutiny, and — in anti-money-laundering contexts — criminal liability for the submitting institution and its officers. Despite these stakes, the tooling landscape available to data governance teams remains fragmented. Manual validation in spreadsheet software does not scale beyond tens of thousands of rows and cannot apply many rules concurrently. Established commercial data-quality platforms typically require six-to-eighteen-month implementation programs, dedicated engineering teams of ten to twenty people, and annual licensing costs that frequently exceed several hundred thousand dollars. Home-built validation scripts, usually written in Python or SQL, tend to be undocumented, non-reusable, and dependent on the individual engineers who authored them. Vendor-managed SaaS alternatives require uploading regulated data to third-party infrastructure, which is frequently unacceptable on data-sovereignty grounds for banking, insurance, and healthcare institutions.

DataGuard was designed to close this gap directly. It is self-hosted, eliminating data-residency concerns; it deploys in minutes rather than months; it is configurable through a browser interface without a dedicated engineering team; and it is AI-accelerated, allowing validation rules to be authored from plain-English descriptions in seconds rather than specification documents that take weeks to produce and review. This paper documents the platform's technical architecture across its five integrated domains — system design, AI integration, regulatory rule encoding, data lineage analysis, and enterprise connectivity — and extends the original architectural description with a worked validation

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

walkthrough, a structured comparison against the four incumbent approaches, and a discussion of the design's limitations and future direction.

1.1 Why a Layered Monolith, Not Microservices

The dominant architectural convention for new enterprise software over the last decade has been microservices: independently deployable services, each with its own datastore, coordinated over a network. That convention optimizes for horizontal scalability and independent team ownership — properties that matter for multi-tenant, internet-scale products. DataGuard's deployment profile is different on both axes: it is installed once per regulated organization (single-tenant), typically inside that organization's own network perimeter (self-hosted), and operated by a small governance or data-quality team rather than a platform engineering organization. Under that profile, the operational cost of a distributed system — service discovery, inter-service authentication, distributed tracing, eventual consistency — is pure overhead. A layered monolith keeps the entire request path inside one auditable process, which matters directly for the regulatory audiences this platform serves: an examiner asking "where does this number come from" is answered by one call stack, not a distributed trace across a dozen services.

1.2 Paper Roadmap

- Section 2 — the four-layer system architecture and the codebase that implements it
- Section 3 — the four Claude-API-based AI subsystems
- Section 4 — the regulatory rule pack engine and its coverage across five sectors
- Section 5 — the cross-source data lineage engine and its three-tier matching algorithm
- Section 6 — the enterprise connectivity layer
- Section 7 — identity, security, and the governance lifecycle
- Section 8 — the Git-based, version-controlled chain of custody
- Section 9 — a worked, end-to-end validation session on the live application
- Section 10 — a structured comparison against manual, commercial, in-house, and SaaS alternatives
- Sections 11–13 — discussion, limitations and future work, and availability

II. SYSTEM ARCHITECTURE

2.1 Layered Monolith Design

DataGuard implements a deliberate layered-monolith architecture rather than a microservices topology. For regulated-industry deployments, this choice prioritizes operational simplicity, data sovereignty, and auditability over the horizontal-scalability benefits of a distributed system — a tradeoff that matches the deployment profile of a self-hosted, single-tenant governance tool. The architecture comprises four layers, illustrated in Figure 1.

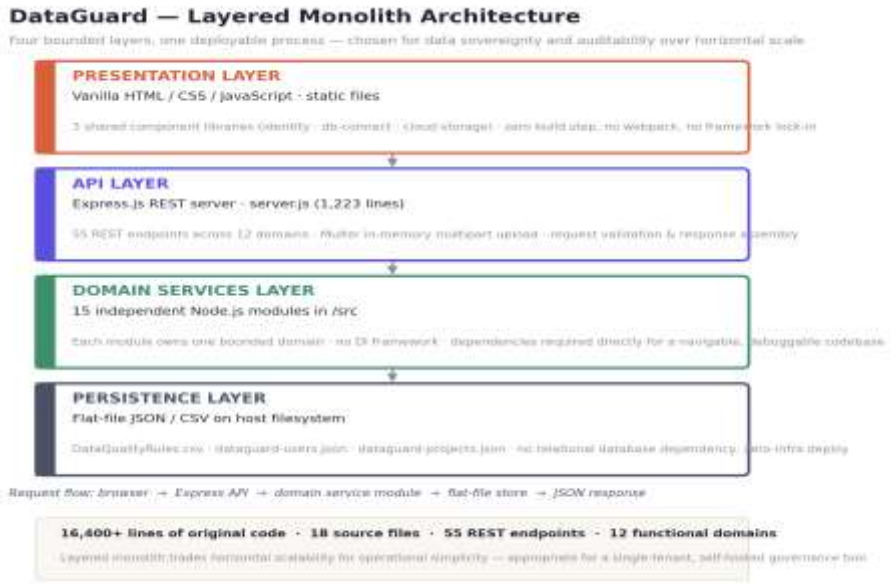


Figure 1. DataGuard's four-layer architecture — presentation, API, domain services, and persistence — with the responsibility and design rationale of each layer.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

The presentation layer is a vanilla HTML/CSS/JavaScript frontend served as static files, with three shared component libraries providing cross-page identity, database-connection, and cloud-storage functionality without any build-step dependency — no webpack, no transpilation, no framework lock-in. This is a deliberate simplicity choice as much as the monolith itself: a regulated-industry buyer's security review can read the frontend's source directly in a browser's developer tools, without a build pipeline standing between the shipped artifact and the reviewed source.

The API layer is an Express.js REST server (1,223 lines) that handles routing, business-logic orchestration, request validation, and multi-part file upload processing; uploaded files are processed via Multer's in-memory storage and are never written to disk. The domain services layer consists of fifteen independent Node.js modules, each owning a single bounded domain with a clean internal interface and no dependency-injection framework, keeping the codebase directly navigable and debuggable. The persistence layer uses flat-file JSON/CSV storage on the host filesystem rather than a relational database dependency, enabling zero-infrastructure deployment — a DataGuard instance can be stood up on a single virtual machine with no database server to provision, patch, or back up separately.

2.2 Codebase Composition

The platform comprises over 16,400 lines of original code across eighteen source files. Table 1 summarizes the principal components.

File	Domain	Lines
server.js	Express.js HTTP server, all API routing	1,223
src/regulatoryRulePacks.js	7 regulatory rule pack definitions across 5 sectors	568
src/lineage.js	Cross-source column lineage engine	472
src/dbConnector.js	8-type database abstraction layer	353
src/ruleManager.js	Rule CRUD engine (CSV-backed)	329
src/gitConnector.js	GitHub/GitLab OAuth + code integration	267
src/fileLoader.js	Multi-format file parser (CSV/JSON/Parquet)	232
src/fraudHeuristics.js	AML pattern detection engine	229
src/validator.js	Core rule-by-rule validation engine	168
public/rules.html	Governance & Validation UI (6 tabs)	2,645
public/validator.html	Main validation interface with charts	2,005
public/lineage.html	Data Lineage UI (7 tabs incl. DocuSmart)	1,668
public/index.html	Public landing page	827

Table 1. Principal source files and their functional domains.

Two patterns are visible in Table 1 that are worth naming explicitly. First, the backend domain modules (server.js plus the nine src/ files) total roughly 3,850 lines against ~7,150 lines of frontend HTML/CSS/JS across four pages — the interface layer is, by line count, the larger half of the system, consistent with a product whose primary users are governance analysts working through a browser rather than engineers integrating via API. Second, the two largest single files after server.js are UI surfaces (rules.html, validator.html) rather than backend logic, reflecting the deliberate choice to keep each UI self-contained in one file rather than decomposed into a component framework — a tradeoff against maintainability at greater scale, discussed further in Section 12.

2.3 REST API Surface

DataGuard exposes fifty-five REST endpoints spanning twelve functional domains: validation (multi-file upload, server-path, and live database validation), rules management (full CRUD plus an AI-powered natural-language rule

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

parser), rule pack lifecycle management, cross-source lineage construction and navigation, AI chat and metadata cataloging, database connection testing and schema introspection, cloud storage listing and schema extraction, multi-modal identity, GitHub/GitLab OAuth 2.0, project lifecycle management, Git repository integration, email notifications, and system health checks. The breadth of this surface reflects the scope of functionality unified within a single deployable application; Appendix A lists the twelve domains against their representative endpoints in full.

III. ARTIFICIAL INTELLIGENCE INTEGRATION

DataGuard integrates Anthropic's Claude API across four functional domains, each requiring distinct prompt engineering, context design, and response-parsing logic. Figure 2 summarizes the four subsystems and their shared dependency on the API.

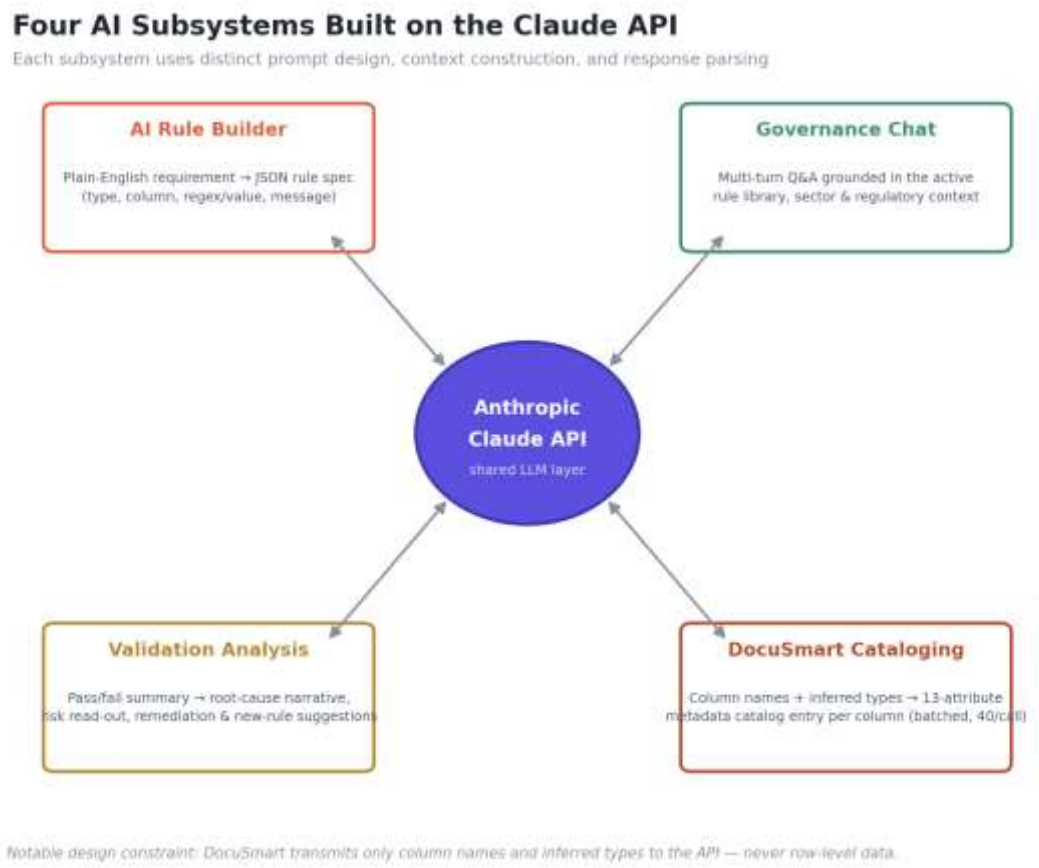


Figure 2. The four Claude-API-based subsystems, each with distinct input, output schema, and parsing logic.

3.1 AI Rule Builder

The AI Rule Builder converts a plain-English validation requirement into an executable rule. For example, the input "The policy number must be between 6 and 20 uppercase alphanumeric characters" is sent to Claude with a structured prompt defining the available rule-type vocabulary, column-name inference logic, and output schema. Claude returns a JSON object specifying the rule type, target column, rule value (e.g., a regular expression), and error message, which the validation engine can execute immediately. This removes the intermediate step in which a data engineer manually interprets a business requirement and writes the corresponding regular expression, enumeration list, or threshold logic by hand — collapsing what is typically a multi-day request-and-review cycle between a compliance analyst and a data engineer into a single browser interaction.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

3.2 AI Governance Chat

The Governance Chat is a multi-turn conversational interface initialized with context about the user's active rule library, industry sector, and applicable regulatory frameworks, allowing questions such as "What BCBS 239 requirements apply to LEI validation?" to receive regulatory-aware, actionable answers rather than generic responses. Because the conversation is grounded in the rule library actually configured for that deployment, answers reference the analyst's own rules by identifier rather than describing regulatory requirements in the abstract.

3.3 AI Validation Analysis

After a validation run, DataGuard can submit a structured summary of results — pass rates, top failing rules, and sample error rows — to Claude for narrative analysis. The model returns a structured report covering probable root causes, regulatory risk assessment, recommended remediation steps, and suggested new rules to catch similar issues going forward, converting raw validation output into actionable governance intelligence rather than a bare pass/fail count.

3.4 DocuSmart: AI Metadata Cataloging Engine

DocuSmart is the platform's most technically involved AI feature. Given column names and inferred data types from any loaded source — files, databases, or cloud storage — it generates a complete business metadata catalog entry per column, comprising thirteen structured attributes: technical name, business name, plain-English description, recommended canonical data type, nullability, PII and PHI (HIPAA) flags, industry-standard identification (e.g., LEI, NPI, ISIN), default value, recommended canonical format, semantic usage category, recommended data-quality rules, and a governance sensitivity classification (public, internal, confidential, or restricted).

The engine batches forty columns per API call to manage context-window constraints, applies a structured prompt that communicates regulatory sensitivity criteria, includes JSON parsing and error-recovery logic, falls back to a rule-based business-name generator when the model returns incomplete data, and exports a standards-compliant CSV data dictionary. Notably, DocuSmart transmits only column names and inferred types to the Anthropic API — never underlying row-level data — preserving data confidentiality for regulated datasets even when the AI feature is active.

IV. REGULATORY RULE PACK ENGINE

DataGuard's seven regulatory rule packs encode practitioner-level regulatory knowledge into executable, field-level validation logic across five industry sectors. Figure 3 and Table 2 summarize each pack's regulatory basis, sector, and rule count.

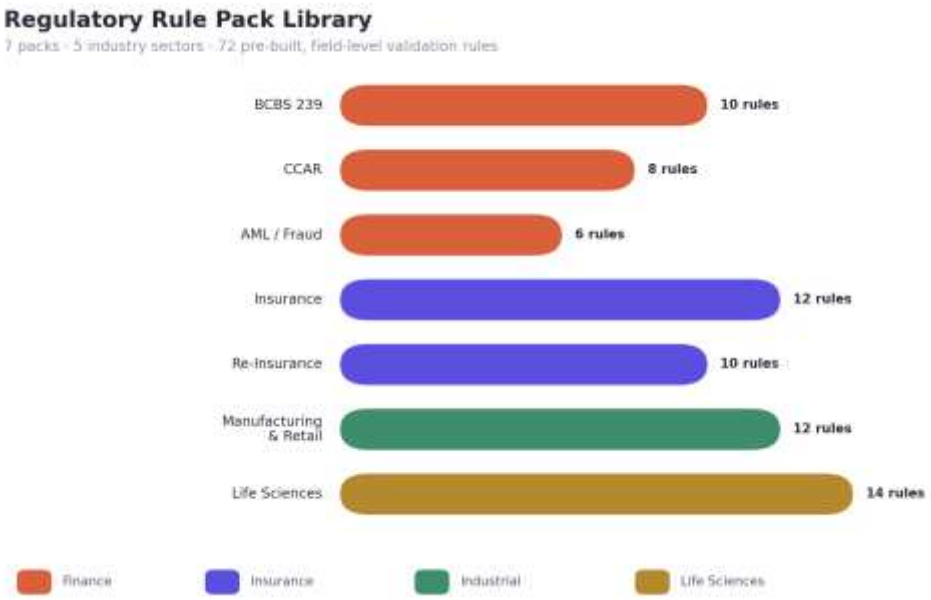


Figure 3. Rule count by pack, colour-coded by industry sector — 72 pre-built field-level rules across seven regulatory frameworks.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

Pack	Regulatory Framework	Sector	Rules
BCBS 239	Principles for Effective Risk Data Aggregation and Risk Reporting	Finance	10
CCAR	Comprehensive Capital Analysis and Review	Finance	8
AML / Fraud	Anti-Money-Laundering Pattern Heuristics	Finance	6
Insurance	Policy, Claims & Policyholder Data Quality	Insurance	12
Re-Insurance	Treaties, Cessions & Reserving	Insurance	10
Manufacturing & Retail	Manufacturing, Retail & Automotive Data Quality	Industrial	12
Life Sciences	Clinical, Pharmacovigilance & Regulatory Submissions	Life Sciences	14

Table 2. Regulatory rule packs, their governing frameworks, and rule counts.

4.1 Depth of Encoded Domain Knowledge

The BCBS 239 pack, for example, encodes checks derived from the Basel Committee's 2013 publication on risk data aggregation, including LEI format validation (ISO 17442), exposure-amount non-negativity, ISO 4217 currency codes, risk-type taxonomy, counterparty and business-line completeness, and data-lineage flagging. The CCAR pack reflects Federal Reserve examiner expectations during capital-planning reviews, including FR Y-14 schedule codes, stress-scenario taxonomy (baseline, adverse, severely adverse), and bank holding company identifiers. The Life Sciences pack encodes HIPAA PHI identification, FDA 21 CFR Part 11 audit-trail fields, ICH E2B adverse-event coding, MedDRA/CTCAE severity grading, and CDISC SDTM identifiers (USUBJID, DOMAIN).

This breadth of encoded domain knowledge — spanning capital markets, insurance, and clinical regulatory submission standards in a single library — reflects sustained, direct exposure to regulatory program delivery across sectors, rather than generic template logic. A generic validation template can check that a field is non-null or numeric; encoding that a re-insurance cession record must reconcile against a named treaty identifier, or that an adverse-event record must carry a MedDRA-coded severity grade, requires the kind of specification knowledge normally held by a compliance program manager rather than a software engineer.

4.2 Scope and Import Model

Consistent with the platform's own in-application guidance, rule packs are field-level data quality templates inspired by industry standards rather than certifications of regulatory compliance: importing a pack gives an organization a fast, reviewable starting point for a screening layer, not a substitute for its own governance framework or legal sign-off. Higher-risk packs — the AML/Fraud heuristics in particular — are not imported by default and require an explicit opt-in, reflecting the greater judgment required to tune fraud-pattern thresholds for a specific institution's risk appetite.

V. CROSS-SOURCE DATA LINEAGE ENGINE

5.1 Column Matching Algorithm

The lineage engine determines which columns across heterogeneous data sources represent the same real-world attribute using a three-tier matching hierarchy with configurable confidence scores, shown in Figure 4. Tier 1, Exact Match (confidence 1.0), applies when column names are identical across sources. Tier 2, Synonym Match (confidence 0.8), applies when column names appear together in a curated, domain-aware vocabulary — for example, ssn and social_security_number, or amount and txn_amount — with separate vocabularies maintained for financial, insurance, and clinical data. Tier 3, Partial Match (confidence 0.5), applies when one column name is a contiguous substring of another (e.g., cust_id and customer_id), and is used only when no exact or synonym match is found.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

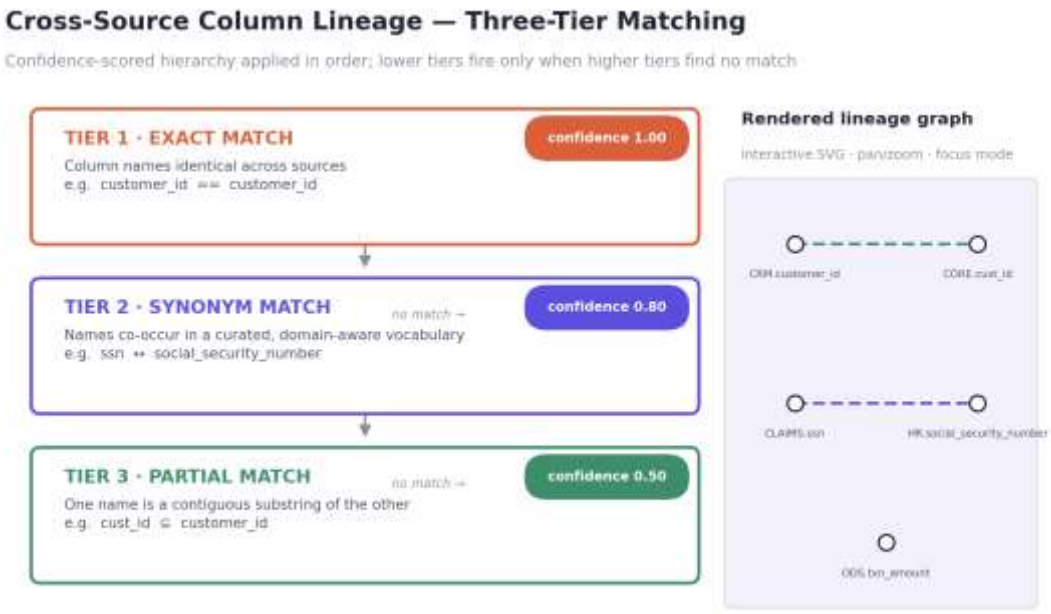


Figure 4. The three-tier column-matching hierarchy (left) and a rendered lineage graph excerpt (right), with edges colour-coded by match tier.

Ordering the tiers this way — attempting the highest-confidence match first and falling through only on failure — avoids a common false-positive failure mode of naive fuzzy-matching lineage tools, where a high-recall similarity score (e.g., edit distance) can wrongly link unrelated columns that happen to share characters. By contrast, tying tier confidence explicitly to the matching mechanism that produced it gives a downstream consumer of the lineage graph — a data steward reviewing a proposed mapping, or the DDL generator described below — a principled basis for deciding which matches to trust automatically and which to review manually.

5.2 Graph Visualization and Derived Outputs

The resulting lineage graph renders as an interactive SVG diagram supporting pan/zoom, bezier-curve edges color-coded by match tier, a focus mode that filters the graph to nodes touching a specific column, and SVG export for regulatory documentation. The graph serves as the foundation for three derived features:

- A DDL generator that produces CREATE TABLE statements across eight database dialects (MySQL, PostgreSQL, Oracle, SQL Server, SQLite, Snowflake, Databricks/Hive, BigQuery) with dialect-aware type mapping.
- A column-mapping export that derives source-to-target mappings from graph edges with a configurable transformation-rule selector (passthrough, rename, cast, concat, split, trim, case conversion, date formatting, hash masking, null-default substitution, or custom SQL), exportable for ETL tool import.
- A feed into the DocuSmart cataloging engine, enabling metadata catalog generation across all connected sources simultaneously.

Taken together, these three outputs mean the lineage graph is not a static diagram but a machine-readable artifact: a data engineer can go from "these five sources probably share a customer identifier" to a reviewed mapping, a dialect-correct CREATE TABLE statement, and a metadata catalog entry, without leaving the same session.

IV. ENTERPRISE CONNECTIVITY LAYER

DataGuard connects to eight enterprise database types — MySQL/MariaDB, PostgreSQL, AWS Redshift, Databricks SQL (via REST API, requiring no JDBC/ODBC dependency), Snowflake, Oracle Database, SAP HANA, and SQLite — three cloud storage providers (AWS S3, Azure Blob Storage, and Google Cloud Storage), and four file formats (CSV/TSV/TXT with auto-detected delimiters, JSON including NDJSON, and Apache Parquet via a pure-JavaScript reader with no native binary dependency), illustrated in Figure 5.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011



Figure 5. The enterprise connectivity layer's three source categories — databases, cloud storage, and file formats — around the DataGuard core engine.

This breadth allows a single platform instance to validate data at rest in a spreadsheet, a cloud data lake, or a production database without requiring a separate ETL step to centralize the data first. For a regulated-industry data governance team, that removes a specific, recurring cost: standing up a temporary extraction pipeline just to run a compliance check against a system that will never otherwise need to leave its source. The Databricks SQL connector is a useful illustration of the platform's general integration approach — rather than adding a JDBC/ODBC driver dependency (which typically requires native binaries and platform-specific packaging), it speaks to Databricks' REST API directly, keeping the connectivity layer within the same pure-JavaScript, zero-native-dependency footprint as the rest of the application.

VII. IDENTITY, SECURITY, AND GOVERNANCE LIFECYCLE

DataGuard implements a four-path, passwordless authentication system: an email/user-ID pairing mode for internal enterprise deployments without SSO; full GitHub OAuth 2.0 authorization-code flow; an equivalent GitLab OAuth 2.0 flow supporting both gitlab.com and self-hosted GitLab instances; and an ephemeral guest mode using session-scoped identifiers that are cleared on tab close or after a fifteen-minute timeout. No passwords are stored anywhere in the system.

The platform's security architecture is designed around data sovereignty as a first-order requirement, as illustrated in Figure 6.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011



Figure 6. The data-sovereignty request lifecycle: an uploaded file is parsed in memory, validated, and discarded — it is never written to disk, logged, or retained.

Uploaded files are processed entirely in memory and are never written to disk, logged, or retained after the response is returned. Database passwords, cloud credentials, and OAuth tokens are used only within the request lifecycle and are never persisted server-side. Secrets are read from environment configuration at startup and never exposed through any API endpoint. A Cloudflare Tunnel provides TLS 1.3 termination, DDoS mitigation, and bot management at the network edge, so the application port is never directly exposed to the internet. As noted in Section 3.4, the AI metadata-cataloging engine transmits only column names and inferred types to the Anthropic API, never row-level data.

A project-based governance lifecycle layer sits above these controls: each named project enforces server-side ownership and sharing controls, links to a sector classification for rule-pack filtering, persists lineage and validator state across sessions, and maintains audit metadata including creation and update timestamps and summary pass/fail counts from the most recent validation run.

VIII. VERSION-CONTROLLED GOVERNANCE CHAIN OF CUSTODY

A distinguishing capability of DataGuard relative to comparable open-source data-quality tools is its Git integration module, which establishes a version-controlled audit trail for governance policy changes — a requirement for enterprise risk and compliance functions under regulatory audit. Through OAuth-authenticated repository access, the platform can:

- List repositories accessible to the connected identity;
- Export the full rule library, in both JSON and CSV form, to a version-controlled subdirectory;
- Commit the live rules file with a timestamped, user-authored commit message; and
- Serialize validation run results to a timestamped results file in the repository.

File writes check for an existing file SHA before choosing between a create or update operation, preserving a clean, non-destructive commit history. In practice, this means the same version-control discipline an engineering team applies to application code — who changed what, when, and why, with a diffable history — is applied to the governance rules themselves, which in most competing tools live only as opaque configuration inside a database.

IX. A WORKED VALIDATION SESSION

To make the architecture described in Sections 2–8 concrete, this section walks through a single, representative session on the live application at dataguard.dataquality.health, following the same demonstration path documented for independent verification: launch the application, load a rule library, and run a validation.

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

9.1 Entry and Identity

A visitor to the landing page is presented with the platform's value proposition — AI-powered field validation, cross-source lineage, and pre-built regulatory rule packs for banking, insurance, life sciences, and supply chain — alongside a compliance-framework checklist (BCBS 239, CCAR, AML, HIPAA, 21 CFR Part 11) and a self-hosted/secure badge. Selecting "Launch DataGuard" presents the four-path identity flow described in Section 7: GitHub OAuth, GitLab OAuth, email/user-ID pairing, or the ephemeral guest mode. A reviewer without an existing account can continue as a guest and reach the full validator interface immediately, with the interface plainly noting that guest projects will not persist between sessions — an explicit, visible tradeoff rather than a silent limitation.

9.2 Validator and Governance Workspace

The Validator workspace presents three elements side by side: a project selector, a drag-and-drop upload zone for CSV, JSON, TXT, and Parquet files, and a settings panel toggling AI Analysis (Section 3.3) and email notification on completion. With a rule library already active — nineteen rules in the session observed for this paper — the interface surfaces the active rule count and the next available rule identifier before a single file is uploaded, so an analyst can confirm which governance policy is live before running anything against real data.

The Governance & Quality workspace, reached from the same top navigation, exposes five tabs: AI Chat (Section 3.2), Add Rule, Rule Packs (Section 4), Validation, and Git/Deploy (Section 8). The AI Chat tab opens with a set of suggested prompts — "Add a rule: Email must contain @", "Disable the SSN format rule", "What fraud heuristics are available?" — that double as a lightweight onboarding aid: a new analyst can see the shape of a valid request before typing their own. The Rule Packs tab presents each pack (Section 4) as a card with its regulatory scope, a rule count, a Preview action, and an Import action, prefaced by an explicit scope note that packs are starting templates rather than compliance certifications — the same caveat given in Section 4.2, surfaced at the point of use rather than only in documentation.

9.3 Data Lineage Workspace

The Data Lineage workspace (Section 5) separates source ingestion — files, databases, or cloud storage, added independently — from graph construction, which is triggered explicitly via a "Build Lineage Graph" action rather than running automatically on upload. This separation matters for large multi-source environments: an analyst can stage five or six sources before committing to the (potentially expensive) cross-source matching pass, rather than re-triggering it after every individual source is added. Once built, the graph view offers filtering by match tier (Exact / Synonym / Partial, Section 5.1), a column-focus search, and three companion views — Column Search, Source Matrix, and Schema Explorer — that expose the same underlying graph from a tabular rather than a visual perspective, for users who prefer to scan a list of proposed mappings rather than read a diagram.

9.4 Observations from the Walkthrough

Three design choices recur across all three workspaces and are worth naming as architectural observations rather than feature descriptions. First, every AI-assisted surface (rule builder, chat, DocuSmart) states its Anthropic API key configuration status inline, rather than failing silently if a key is absent — a small detail, but one that matters for a self-hosted tool an operator configures once and revisits infrequently. Second, every workspace shows the current governance state (active rule count, rules CSV path, next rule identifier) before any action is taken, consistent with the auditability goal stated in Section 1.1. Third, the guest-mode and scope-note disclosures observed in Sections 9.1 and 9.2 indicate a deliberate pattern of surfacing the platform's own limitations at the point of use, rather than only in supporting documentation — a pattern consistent with a tool built for a compliance-literate audience.

X. COMPARATIVE POSITIONING

Section 1 named four incumbent approaches to regulatory data validation. Table 3 restates their characteristics against DataGuard's, using the figures already given in Section 1 and the architectural facts established in Sections 2–8, to make the comparison explicit rather than leaving it implicit in the surrounding prose.

International Journal of Emerging Trends in Engineering and Management Research (IJETEMR)

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

Approach	Typical time to first validation	Data leaves the organization?	Regulatory logic reusable?	Audit trail for rule changes?
Manual spreadsheet checks	Immediate, but does not scale past ~10^4 rows	No	No — ad hoc per analyst	No
Commercial data-quality suite	6–18 months	Depends on deployment model	Yes, via vendor configuration	Vendor-dependent
In-house scripts	Days to weeks per rule set	No	Rarely — tied to the author	Only if separately built
Vendor-hosted SaaS	Days to weeks	Yes — uploaded to third-party infrastructure	Yes, via vendor configuration	Vendor-dependent
DataGuard	Minutes (self-hosted, browser-configured)	No — self-hosted, in-memory processing	Yes — 7 rule packs, AI-authored custom rules	Yes — Git-based (Section 8)

Table 3. DataGuard positioned against the four incumbent approaches named in Section 1, on the dimensions those approaches most commonly fail.

The comparison in Table 3 also clarifies where DataGuard's architectural choices are concessions rather than pure wins. A commercial suite's multi-month implementation typically buys dedicated vendor support, formal SLAs, and — in many cases — a broader connector ecosystem than any single self-hosted tool can match on day one. DataGuard's positioning is specifically for the organization that needs a self-hosted, rapidly deployable screening layer under its own data-sovereignty constraints, not a claim of universal superiority over every dimension a mature commercial platform offers.

XI. DISCUSSION

DataGuard's architectural contribution lies less in any single component — each of AI-assisted rule authoring, database connectivity, and data lineage analysis has precedent elsewhere — and more in their integration into a single, self-hosted, zero-infrastructure-dependency deployable application. Commercial data-quality suites typically achieve comparable functional breadth only through lengthy implementation programs and dedicated engineering teams; open-source alternatives typically address a single domain (e.g., lineage visualization, or rule-based validation) without the combination of regulatory rule packs, LLM-assisted rule generation, and multi-source connectivity described here. The choice of a layered-monolith rather than microservices architecture is a deliberate concession of horizontal scalability in favor of operational simplicity and auditability, appropriate to the single-tenant, self-hosted deployment model the platform targets.

The platform's regulatory rule packs are, by design, encodings of domain expertise rather than purely generic validation templates; their construction required translating specific, practitioner-level regulatory expectations (e.g., FR Y-14 schedule structure, Basel risk-data aggregation principles, CDISC SDTM conventions) into field-level executable logic, which is the central engineering challenge the rule-pack architecture was designed to solve.

XII. LIMITATIONS AND FUTURE WORK

12.1 Architectural Tradeoffs

The design choices documented in this paper carry costs that are worth stating directly rather than leaving implicit. The layered-monolith architecture (Section 2.1) trades away independent scaling of the validation engine versus the API layer — a workload spike in one does not currently scale without scaling the whole process. Flat-file JSON/CSV persistence (Section 2.1) is well suited to a single-instance, single-tenant deployment, but does not provide the

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

concurrent-write guarantees a relational database offers, which would matter for a future multi-user, high-concurrency deployment mode. The frontend's single-file-per-page structure (Section 2.2) keeps the presentation layer legible today, at the cost of the modularity a component framework would offer as the UI surface continues to grow.

12.2 Dependence on a Third-Party LLM Provider

All four AI subsystems (Section 3) depend on the availability, pricing, and output consistency of the Anthropic Claude API. This is a considered dependency, not an oversight — Section 3.4 shows the platform deliberately minimizing what it sends to that API — but it means the AI-accelerated capabilities that most differentiate DataGuard from a purely rule-based validator are unavailable during an API outage, and the DocuSmart and AI Rule Builder outputs, like any LLM output, warrant human review rather than blind execution, a expectation the interface reinforces through the preview-before-import pattern noted in Section 9.2.

12.3 Directions for Future Work

- An optional pluggable persistence backend (e.g., SQLite or PostgreSQL) for deployments that need concurrent multi-user write access, while preserving the zero-infrastructure flat-file mode as the default.
- Extending the three-tier lineage matcher (Section 5.1) with an optional, human-reviewed fourth tier using embedding-based semantic similarity for column names that share no lexical overlap — e.g., matching a source's dob to a target's date_of_birth without a curated synonym entry.
- A model-agnostic abstraction over the four AI subsystems, allowing a self-hosted or alternative LLM provider to be substituted where an organization's policy precludes any third-party API call, even one limited to column metadata.
- Formal, independent regulatory-compliance review of individual rule packs by domain specialists in each covered sector, to complement the in-application scope disclaimer described in Section 4.2.

XIII. AVAILABILITY

The claims in this paper are independently verifiable. The live platform is available at dataguard.dataquality.health without login required for the landing page and feature demonstration; the validator interface, governance and rules module, and data lineage module are reachable from the platform navigation, and a guest-mode session (Section 9.1) reaches the full interface without account creation. Version-controlled source code with full commit history is available at github.com/ntyagi29/dataguard-agent. The version described in this paper is DataGuard v2.9.0.

XIV. CONCLUSION

DataGuard integrates large-language-model-assisted rule authoring, an eight-database and three-cloud-provider connectivity layer, a three-tier cross-source lineage inference algorithm, and a library of seven regulatory rule packs spanning five industry sectors into a single self-hosted, zero-infrastructure-dependency application. Its layered-monolith architecture, passwordless multi-modal identity system, in-memory file handling, and version-controlled Git-based governance chain of custody are designed around the specific constraints of regulated-industry deployment: data sovereignty, auditability, and rapid, low-overhead implementation. The worked walkthrough in Section 9 and the comparative positioning in Section 10 are offered to make those design claims checkable rather than asserted; the limitations named in Section 12 are offered in the same spirit. We hope this architectural description is useful to practitioners and researchers evaluating or building similar systems for regulatory data quality and governance.

REFERENCES

[1] Basel Committee on Banking Supervision. (2013). Principles for Effective Risk Data Aggregation and Risk Reporting (BCBS 239). Bank for International Settlements.
[2] Board of Governors of the Federal Reserve System. Comprehensive Capital Analysis and Review (CCAR): Program overview and FR Y-14 reporting instructions.
[3] Financial Crimes Enforcement Network (FinCEN). Anti-Money-Laundering (AML) program requirements, 31 U.S.C. § 5311 et seq.
[4] U.S. Department of Health and Human Services. Health Insurance Portability and Accountability Act of 1996 (HIPAA), Privacy and Security Rules.
[5] U.S. Food and Drug Administration. Title 21 CFR Part 11 — Electronic Records; Electronic Signatures.
[6] International Organization for Standardization. ISO 17442:2020 — Legal Entity Identifier (LEI) code.
[7] International Organization for Standardization. ISO 4217 — Currency codes.

|ISSN: 2455-7773 | www.ijetemr.com| A Double-Blind Peer-Reviewed and Bimonthly Scholarly Journal|

|Volume 11, Issue 3, May-June 2026|

DOI:10.15662/ijetemr.2026.1103011

[8] International Council for Harmonisation. ICH E2B(R3) — Electronic Transmission of Individual Case Safety Reports.
[9] MedDRA Maintenance and Support Services Organization / U.S. National Cancer Institute. MedDRA and CTCAE severity grading terminologies.
[10] Clinical Data Interchange Standards Consortium (CDISC). Study Data Tabulation Model (SDTM) Implementation Guide.

APPENDIX A — REST API DOMAIN SUMMARY

The fifty-five REST endpoints described in Section 2.3 are grouped below by functional domain, with a representative endpoint purpose for each.

#	Functional Domain	Representative Capability
1	Validation	Multi-file upload, server-path, and live database validation runs
2	Rules Management	Full CRUD plus AI-powered natural-language rule parsing
3	Rule Pack Lifecycle	Preview, import, and manage the 7 regulatory rule packs
4	Lineage Construction	Build and navigate the cross-source column lineage graph
5	AI Chat & Cataloging	Governance chat and DocuSmart metadata catalog generation
6	Database Connectivity	Connection testing and schema introspection across 8 engines
7	Cloud Storage	Listing and schema extraction across 3 cloud providers
8	Identity	Multi-modal, passwordless authentication (Section 7)
9	OAuth	GitHub / GitLab OAuth 2.0 authorization-code flows
10	Project Lifecycle	Create, share, and manage governance projects
11	Git Integration	Repository listing, rule export, and commit operations (Section 8)
12	Notifications & Health	Email notifications and system health checks

Table 4. The twelve REST API functional domains referenced in Section 2.3.

APPENDIX B — GLOSSARY

AML. Anti-Money-Laundering — regulatory obligations to detect and report suspicious financial activity.
BCBS 239. Basel Committee principles for effective risk data aggregation and risk reporting.
CCAR. Comprehensive Capital Analysis and Review — the Federal Reserve's annual bank stress-testing and capital-planning program.
CDISC SDTM. Clinical Data Interchange Standards Consortium's Study Data Tabulation Model, a standard for clinical trial data structure.
DocuSmart. DataGuard's AI-powered metadata cataloging engine (Section 3.4).
LEI. Legal Entity Identifier — a 20-character alphanumeric code (ISO 17442) uniquely identifying a legal entity in financial transactions.
Lineage tier. One of three confidence levels (Exact, Synonym, Partial) assigned to a proposed cross-source column match (Section 5.1).
Rule pack. A pre-built, importable library of field-level validation rules mapped to a specific regulatory framework and sector (Section 4).
Layered monolith. A single deployable application internally organized into strict layers, as opposed to a distributed microservices topology (Section 2.1).