

DATA SLAs THAT HOLD: ENGINEERING CONTRACT-GRADE RELIABILITY BETWEEN DATA PRODUCERS AND CONSUMERS

Santosh Kumar Maddali

Sr. Data Engineer

ABSTRACT

Modern enterprises run on data pipelines whose outputs feed regulatory reporting, financial close, operational analytics, and customer-facing products, yet the reliability guarantees attached to those pipelines rarely rise above informal expectations. When an upstream team renames a column, silently changes a unit of measure, or delays a partition, the failure typically surfaces far downstream, inside a broken dashboard, a corrupted derived dataset, or a misstated financial figure, long after the causal change occurred. This paper argues that the discipline that transformed service reliability in the 2010s, namely the Service Level Indicator/Objective/Agreement (SLI/SLO/SLA) framework of Site Reliability Engineering, can and should be transposed onto data exchange, but only if it is coupled to an enforcement substrate the software world calls a contract. We develop a taxonomy of data-specific SLIs (freshness, completeness, validity, consistency, schema stability, and availability), define what elevates a data SLA from an aspirational document to a contract-grade instrument, and describe a reference architecture (contract registry, CI-time compatibility gates, write-audit-publish promotion, continuous SLI computation, error budgets, and automated quarantine) that makes violations mechanically detectable and attributable before they propagate. We further examine the organizational mechanisms (asset tiering, consumer-driven negotiation, producer on-call, and deprecation policy) without which the technical machinery decays, and we catalogue recurring anti-patterns reported in the practitioner and research literature. The contribution is a coherent, implementation-oriented synthesis that treats data reliability as an engineering discipline with measurable objectives and enforceable obligations rather than a matter of goodwill between teams.

Keywords:

Data Contracts; Service Level Agreements; Data Reliability Engineering; Data Quality; Schema Evolution; Data Observability; Data Mesh; Error Budgets

I. INTRODUCTION

The consequences of unreliable data are no longer confined to analytics teams. Data products now sit on the critical path of revenue recognition, credit decisioning, fraud detection, personalization, and enterprise forecasting. Sculley et al., studying large production systems, identified data dependencies as one of the most expensive and least visible forms of technical debt, noting that data dependencies cost more than code dependencies precisely because they are unstable, underspecified, and rarely versioned [1]. The observation generalizes well beyond the systems they studied: any consumer computing on inputs it does not control inherits risk from every unannounced upstream change.

Software engineering confronted an analogous problem a decade earlier. Distributed services with informal reliability expectations produced cascading outages and inter-team blame until Google's Site Reliability Engineering practice reframed reliability as a measurable, budgeted quantity: define an indicator (SLI), commit to an objective (SLO), and, where money or obligation is at stake, bind it in an agreement (SLA) with explicit consequences [2], [3]. The framework worked because it converted an ambient anxiety ('the service should be reliable') into an engineering artifact with a number, an owner, and a feedback loop.

Data exchange between teams has not undergone the same transformation. Datasets are consumed through implicit interfaces such as a table someone found in the warehouse or a topic someone subscribed to, with no declared schema semantics, no committed freshness, and no notification channel for change. Kleppmann observes that in large organizations different teams inevitably evolve at different speeds, making explicit compatibility guarantees on data encodings a precondition for independent evolution [4]. The data mesh movement formalizes the organizational half of the answer: data must be treated as a product, owned by the domain that produces it, with consumers treated as customers [5]. What remains underspecified in much of the literature is the mechanical half: what exactly must be measured, promised, and enforced for a data SLA to hold in the sense that a load-bearing structure holds: predictably, under stress, with graceful and attributable failure.

This paper addresses that gap. The work is a conceptual synthesis: it derives a framework from the published literature on data quality, distributed systems, and service reliability, and it presents no implemented system, proprietary architecture, or empirical evaluation. All parameters, thresholds, and objectives appearing in the tables and figures are illustrative values chosen to demonstrate the form a contract term takes, not measurements. Our contributions are fourfold: (i) a taxonomy of data SLIs with precise, computable definitions suited to batch and streaming delivery (Section III); (ii) a definition of contract-grade reliability and a specification of the data contract as its enforcement substrate (Section IV); (iii) a reference architecture that operationalizes detection, prevention, and attribution of contract violations (Section V); and (iv) an account of the organizational mechanisms and anti-patterns that determine whether the technical machinery survives contact with real teams (Sections VI-VII). The synthesis draws on three established bodies of work: data-quality measurement, distributed data-system reliability, and software service-level engineering. From each, concepts are drawn where they supply a measurable mechanism for defining, enforcing, or governing producer-consumer reliability obligations, and are then mapped onto the SLI taxonomy, contract specification, and operational architecture developed in the following sections.

II. BACKGROUND AND RELATED WORK

Data quality research long predates the current tooling wave. Wang and Strong established empirically that quality is consumer-relative, a multidimensional construct spanning intrinsic, contextual, representational, and accessibility dimensions, implying that quality targets cannot be set by producers in isolation [6]. Batini et al. surveyed assessment and improvement methodologies and highlighted the persistent difficulty of connecting measured quality dimensions to operational processes that improve them [7]. The SLA framing supplies exactly that connection: a dimension becomes an SLI, a target becomes an SLO, and the improvement process becomes error-budget-driven prioritization.

On the systems side, Schelter et al. described Amazon's Deequ, which compiles declarative quality constraints (completeness, uniqueness, value distributions) into efficient computations over large datasets, demonstrating that constraint verification can run at production scale as a routine pipeline stage rather than an occasional audit [8]. Modern transactional table formats demonstrate that atomic, versioned publication of table states over object storage is achievable at production scale [9], which matters here because atomic publication is what makes 'the dataset either satisfies the contract or is never exposed as fulfilled' implementable. Kreps's log-centric architecture supplies the streaming analogue: an immutable, ordered log as the interchange point at which schema and semantics can be pinned [10].

From software architecture, this work borrows the consumer-driven contract pattern articulated by Robinson, in which consumers express the subset of a provider's interface they actually depend on, and the provider validates changes against the union of those expectations [11]. Our synthesis stitches these threads (quality dimensions, scalable verification, transactional publication, consumer-driven contracts, and SRE's budgeting discipline) into a single operating model.

III. FROM SOFTWARE SLAs TO DATA SLAs: A TAXONOMY OF INDICATORS

An SLI is only useful if it is precisely computable from telemetry, robust to gaming, and meaningful to the consumer's use case. Naive transpositions from services fail: 'uptime' of a warehouse says nothing about whether yesterday's partition is complete, and 'latency' of a query says nothing about whether the data queried is stale. We therefore define six data-native indicator families, summarized in Table I. Figure 1 situates the taxonomy within the SLI-SLO-SLA hierarchy that structures the remainder of the paper: indicators are computed continuously from telemetry, objectives bind them to targets over declared windows, and the agreement layer attaches consequences and bilateral obligations through the data contract.

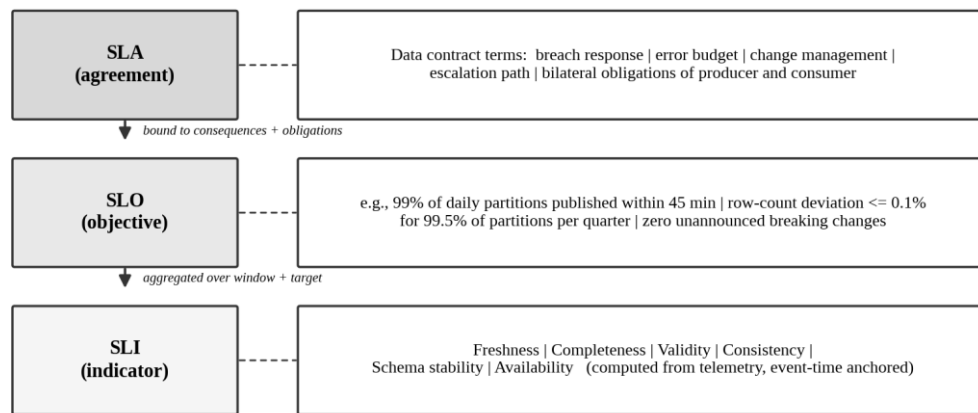


Fig. 1. The SLI-SLO-SLA hierarchy instantiated for data exchange
Table 1. Data SLI taxonomy with computable definitions and illustrative SLOs

SLI Dimension	Definition	Example Measurement	Example SLO
Freshness	Elapsed time between event occurrence (or source commit) and availability of the corresponding record in the consumable dataset	Batch: publish_time - expected_source_close_time per partition. Streaming: p99(availability_time - event_time) over eligible records, reported with watermark coverage	99% of daily partitions available within 45 minutes of source close-of-book
Completeness	Fraction of expected records or entities present relative to an authoritative expectation (source counts, forecasts, or reconciliation control totals)	row_count / expected_count per partition; entity coverage against reference dimension	Row-count deviation from reconciled source total <= 0.1% for 99.5% of partitions per quarter
Validity	Conformance of individual values to declared types, domains, ranges, and referential constraints	Share of rows passing declarative constraint suites (non-null, uniqueness, foreign-key, range checks)	>= 99.99% of rows satisfy all Tier-1 constraints; zero primary-key duplicates
Consistency	Agreement of derived aggregates with upstream systems of record and with sibling datasets computed from the same lineage	Cross-system reconciliation deltas on control metrics (e.g., revenue, order counts)	Daily reconciliation delta <= 0.05% on all control metrics
Schema stability	Absence of breaking structural or semantic change relative to the published contract version	Count of incompatible schema mutations detected at the registry or CI gate	Zero unannounced breaking changes; >= 30-day deprecation window for announced ones
Availability	Probability that the dataset endpoint (table, topic, API) is queryable and serving the contracted version	Successful synthetic reads / total synthetic reads over rolling 28 days	99.9% monthly read availability for Tier-1 assets

Three design principles govern the taxonomy. First, event-time anchoring: freshness must be measured against event time, not processing time, or a pipeline that drops late-arriving data will appear paradoxically 'fresher' as it becomes less complete. Freshness and completeness are duals and must be objectified jointly; for example, 'partition is published within 45 minutes and contains $\geq 99.9\%$ of records ultimately attributable to it after a 7-day late-arrival horizon.' Second, expectation externality: completeness requires an expectation independent of the pipeline being measured (source-system control totals, reconciliation counts, or a statistically fitted forecast with declared tolerance), otherwise the pipeline grades its own homework. Third, percentile discipline inherited from SRE: objectives are stated over a population and a window (e.g., 99.5% of partitions per rolling quarter), never as absolutes, because absolute promises are either trivially violated or drive teams to hide failures [2], [3]. Schema stability deserves special treatment because it is the dimension through which most catastrophic violations arrive. A breaking change is not merely a removed column or a narrowed type; it includes semantic breaks invisible to structural checkers, such as a currency field silently switching from cents to dollars, a status enum acquiring a new value that downstream CASE logic maps to NULL, or a timezone convention changing. Contract-grade schema stability therefore requires the contract to carry semantics (units, enumeration domains, nullability intent, identifier stability) alongside structure, so that semantic mutations are also declarable, detectable, and versionable [4], [11].

IV. DATA CONTRACTS AS THE ENFORCEMENT SUBSTRATE

We define a data SLA as contract-grade when it satisfies five properties: (1) Specificity: every promised behavior corresponds to a computable SLI with a declared measurement method and window; (2) Bilaterality: obligations bind both parties, the producer to the SLOs and change-management terms, the consumer to declared usage (fields depended upon, load patterns, and a registered point of contact); (3) Machine-enforceability: the contract exists as a versioned artifact that automated systems consult to admit or reject changes and data; (4) Attributability: every violation is traceable to a responsible interface version and owning team within bounded time; and (5) Consequence: breaches trigger predefined responses (error-budget consumption, freeze of risky changes, escalation), not ad-hoc negotiation.

The concrete instrument is the data contract: a version-controlled specification attached to a data product's output port. A complete contract carries:

- Structural schema: fields, types, nullability, keys, partitioning, and delivery mode (batch table, stream topic, API).
- Semantic layer: units, enumeration domains with closed/open declarations, identifier stability guarantees, timezone and encoding conventions, and privacy classification of each field.
- Reliability terms: the SLOs of Section III with measurement definitions, plus the late-arrival horizon and reprocessing (backfill) semantics, that is, whether history is immutable, append-only, or subject to restatement, and how restatements are signaled.
- Change management: compatibility mode (e.g., backward-transitive), deprecation window, notification channel, and the migration obligations of each side.
- Operational metadata: owning team, on-call rotation, escalation path, tier classification, and links to lineage and runbooks.

Crucially, contracts should be consumer-driven where possible [11]. When each consumer registers the projection of the interface it actually uses, the producer gains a precise blast-radius map: a proposed change is validated against the union of registered expectations, and a change that breaks no registered consumer can proceed on an expedited path. This converts the classic standoff (producers afraid to change anything, consumers afraid of everything) into a tractable dependency-resolution problem, and it gives the deprecation process a definitive completion signal: the last registered dependency on a deprecated field disappearing.

V. REFERENCE ARCHITECTURE FOR CONTRACT-GRADE RELIABILITY

Figure 2 depicts the reference architecture. Its components divide into a prevention path (registry and CI gates), an admission path (write-audit-publish with quarantine), and an independent detection-and-response plane (continuous SLI computation, error budgets, and escalation). Each is described in turn.

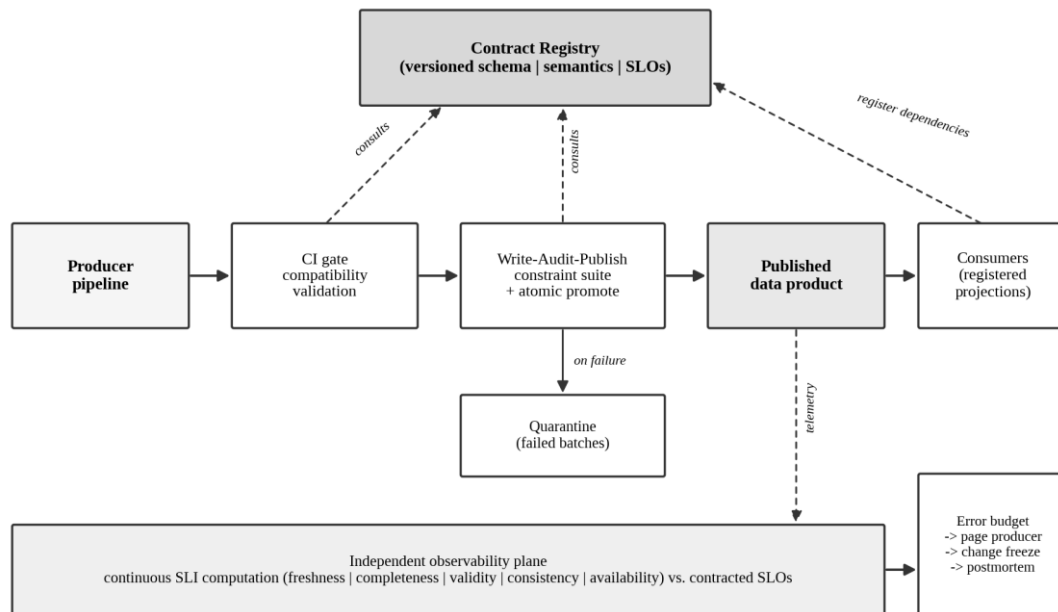


Fig. 2. Reference architecture for contract-grade data reliability

A. Contract Registry and Version Control

Contracts live in version control adjacent to producer code and are compiled into a central registry that exposes them to CI systems, orchestrators, catalogs, and monitors. The registry is the single source of truth for 'what is promised'; every downstream enforcement point derives from it. Contract changes follow the same review workflow as code, with consumer representatives as required reviewers for Tier-1 assets.

B. Prevention: CI-Time Compatibility Gates

The cheapest violation is the one that never ships. Producer CI pipelines validate every proposed schema and semantic change against the registry's compatibility rules and the registered consumer projections before merge. Streaming interchange points enforce the same rules at the schema-registry layer, rejecting incompatible writer schemas at publish time [4], [10]. Semantic checks (unit annotations, enum domain diffs) run alongside structural ones, which is precisely why semantics must be first-class in the contract rather than prose in a wiki.

C. Admission: Write-Audit-Publish

Contract-grade delivery requires that data failing its contract never becomes visible as fulfilled data. The write-audit-publish pattern stages each batch or micro-batch in a non-consumable location, executes the contract's constraint suite against it, using scalable declarative verification of the kind demonstrated by Deequ [8], and atomically promotes it only on success. Transactional table formats make the promotion step atomic and versioned, enabling three clean failure postures: block (Tier-1 default; freshness SLI degrades but validity never does), quarantine-and-publish-partial with explicit flagging, or publish-with-annotation for advisory checks. The choice of posture per constraint is itself a contract term, negotiated per tier.

D. Detection: Continuous SLI Computation and Observability

An independent observability plane computes every contracted SLI from telemetry (publish events, event-time watermarks, row counts, constraint outcomes, reconciliation deltas, and synthetic reads) and evaluates them against SLOs on their declared windows. Independence matters: the measurement system must not share failure modes with the pipeline it measures, echoing the SRE separation between serving and monitoring [2]. Beyond contracted SLIs, distributional monitoring (drift in value distributions, null rates, and cardinalities against historical baselines) provides early warning of semantic breaks that enumerated constraint suites cannot anticipate.

E. Budgeting and Response: Error Budgets, Quarantine, and Incident Discipline

Each SLO implies an error budget, the tolerated shortfall over the window [3]. Budget consumption, not raw alert volume, drives response: a Tier-1 asset that has burned half its quarterly freshness budget in a week triggers a change freeze on its upstream pipeline and a producer-led review, exactly as SRE practice freezes risky launches on exhausted budgets. Violations page the producer, not the consumer; this inversion is what distinguishes a contract regime from the status quo, in which consumers discover producer failures. Downstream, consumers of

contracted assets can implement circuit breakers: a reporting or serving system that detects an upstream freshness or validity breach falls back to its last known-good snapshot rather than silently consuming compromised inputs. Every Tier-1 breach concludes with a blameless postmortem whose action items amend either the pipeline or the contract; a contract that is repeatedly breached and never amended is a fiction, and the review process is what keeps the paper synchronized with reality.

VI. ORGANIZATIONAL MECHANISMS

Technical machinery decays without matching organizational structure. Four organizational mechanisms follow from the framework developed above.

Tiering. Uniform rigor is unaffordable and unnecessary. Assets are classified by consequence-of-failure, with contract rigor and operational posture scaled accordingly (Table II). Tiering also disciplines negotiation: a consumer requesting Tier-1 terms must justify Tier-1 consequences, and the producer's costs become visible and chargeable.

Table II. Asset tiering and corresponding contract rigor

Tier	Consumer Profile	Contract Rigor	Operational Posture
Tier 1	Regulatory reporting, financial close, customer-facing features, real-time decisioning systems	Full contract: schema, semantics, SLOs on all six dimensions, signed change-management terms	24x7 producer on-call, paging alerts, error budgets, post-incident review
Tier 2	Executive dashboards, operational analytics, scheduled forecasting and planning jobs	Contract with SLOs on freshness, completeness, validity, and schema stability	Business-hours support, ticketed alerting, quarterly SLO review
Tier 3	Exploration, ad-hoc analysis, prototyping	Schema declaration and lineage registration only; best-effort quality	No paging; self-service diagnostics via catalog and observability tooling

Negotiation, not decree. SLOs imposed unilaterally by platform teams are ignored; SLOs demanded unilaterally by consumers are refused. Contract terms emerge from a structured negotiation in which consumers state use-case requirements (a fraud-detection system needs 15-minute freshness; a monthly report does not) and producers state feasibility and cost, consistent with the consumer-relative nature of quality established by Wang and Strong [6]. Producer accountability with product framing. The owning domain team carries the pager for its data products and staffs a rotation, per the data-as-a-product principle. In exchange, the platform supplies the registry, verification, and observability infrastructure as paved-road tooling, so that each team does not rebuild the enforcement substrate.

Deprecation policy as a first-class citizen. The most common producer objection to contracts, 'we will never be able to change anything again', is answered by a rigorous deprecation protocol: announce on the contracted channel, dual-publish old and new versions through the deprecation window, monitor registered consumer migration, and retire only when dependencies reach zero. Contracts, properly implemented, increase evolution velocity because producers finally know who consumes what [11].

VII. DISCUSSION: ANTI-PATTERNS AND LIMITATIONS

Several failure modes follow predictably when one of the three conditions above is absent. The aspirational SLA: objectives published without an independent measurement plane, which decay into shelfware. Vanity SLIs: measuring processing-time freshness or self-referential completeness, which reward pipelines for dropping data. The unfunded mandate: declaring producers accountable without giving them paved-road enforcement tooling or headcount for on-call, which converts the program into resentment. Contract sprawl: contracting Tier-3 exploratory assets, which taxes iteration speed and discredits the program. Silent restatement: backfills that rewrite history without signaling, which corrupts downstream incremental consumers even when every per-partition SLO is met; restatement semantics must be contracted explicitly. Finally, blame-oriented enforcement: using SLI dashboards as ammunition rather than budgets as prioritization signals, which drives teams to game measurements, the pathology SRE's blameless-postmortem culture exists to prevent [2], [3].

Limitations of the framework should be stated plainly. Accuracy in the deepest sense, the correspondence of recorded values to real-world truth, is only partially capturable by SLIs; reconciliation and validity are proxies,

and some error classes are detectable only by human domain review. Statistical expectations for completeness carry false-positive costs that must be tuned per asset. And the framework presumes an organization capable of sustaining cross-team agreements; in low-trust environments, the contract registry can become another artifact of theater. These are boundaries of the method, not refutations of it: within its domain (the mechanical detection, prevention, and attribution of interface violations between data teams), the approach converts a chronic source of silent failure into an engineering problem with known solutions.

VIII. CONCLUSION AND FUTURE WORK

Data SLAs hold when three conditions are met simultaneously: the promises are computable (a precise SLI taxonomy anchored in event time and external expectations), the promises are enforceable (a machine-readable contract consulted by CI gates, admission control, and independent observability), and the promises are governed (tiering, negotiation, error budgets, producer on-call, and disciplined deprecation). Remove any leg and the structure reverts to the familiar equilibrium of silent breakage and downstream blame. The synthesis presented here is deliberately implementation-oriented; every component maps to demonstrated systems techniques: declarative verification at scale [8], transactional publication, and schema-registry compatibility enforcement [4], [10], all organized by the budgeting discipline that made service reliability tractable [2], [3].

Future work proceeds in three directions. First, cost-aware SLO optimization: freshness, completeness, and compute cost trade against one another, and the selection of objectives is amenable to formal optimization given consumer utility functions. Second, contract inference: mining query logs and lineage to propose consumer projections and candidate constraints automatically, lowering adoption cost. Third, cross-organizational contracts: extending the enforcement substrate across company boundaries, where data-sharing agreements today remain legal documents with no mechanical teeth, precisely the condition inside organizations that this paper set out to remedy.

REFERENCES

- [1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, Montreal, Canada, 2015, pp. 2503-2511.
- [2] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, 2016.
- [3] B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, *The Site Reliability Workbook: Practical Ways to Implement SRE*. Sebastopol, CA, USA: O'Reilly Media, 2018.
- [4] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, CA, USA: O'Reilly Media, 2017.
- [5] Z. Dehghani, "How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh," *martinfowler.com*, May 2019. [Online]. Available: <https://martinfowler.com/articles/data-monolith-to-mesh.html>
- [6] R. Y. Wang and D. M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," *Journal of Management Information Systems*, vol. 12, no. 4, pp. 5-33, 1996.
- [7] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for Data Quality Assessment and Improvement," *ACM Computing Surveys*, vol. 41, no. 3, Art. 16, 2009.
- [8] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating Large-Scale Data Quality Verification," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781-1794, 2018.
- [9] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Luszczak, M. Switakowski, M. Szafranski, X. Li, T. Ueshin, M. Mokhtar, P. Boncz, A. Ghodsi, S. Paranjpye, P. Senster, R. Xin, and M. Zaharia, "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411-3424, 2020.
- [10] J. Kreps, "The Log: What Every Software Engineer Should Know About Real-Time Data's Unifying Abstraction," *LinkedIn Engineering Blog*, Dec. 2013. [Online]. Available: <https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>
- [11] I. Robinson, "Consumer-Driven Contracts: A Service Evolution Pattern," *martinfowler.com*, Jun. 2006. [Online]. Available: <https://martinfowler.com/articles/consumerDrivenContracts.html>