International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Implementing Zero-Trust Authentication and Authorization in Distributed Data Lake House Architectures

Surajit Paul

Abstract: Distributed data lake house platforms connect business intelligence tools, distributed query engines, metadata catalogs, orchestration frameworks, and cloud object storage into a single analytical surface that spans hybrid and multi-cloud infrastructure. This disaggregation removes the network perimeter that earlier access control models assumed, and it leaves authentication and authorization scattered across components that each enforce a different native model: identity providers issue tokens, query engines apply engine-level roles, metadata catalogs hold table-level ownership, and storage systems apply bucket-level policies. No existing zero-trust reference model maps these control points onto a single, continuously verified architecture for a disaggregated lake house stack. This paper proposes a reference architecture that anchors fine-grained authorization at the metadata catalog, where the platform first has full semantic visibility into which tables, columns, and rows a request may reach, while identity federation and workload identity manage authentication upstream and downstream of that decision point. The architecture is evaluated against zero-trust tenets and compared with engine-native and storage-native enforcement alternatives on the basis of granularity, cross-engine consistency, and auditability. The paper also formalizes the latency cost of continuous verification and a composite behavioral risk score for detecting mid-query trust decay, and it closes with a discussion of the approach's limitations and the conditions under which catalog-anchored enforcement is the stronger design choice.

Keywords: *Access Control, Data Lake House, Fine-Grained Authorization, Identity Federation, Zero-Trust Architecture*

1. Introduction

A firewall used to be worth defending, back when a network boundary meant something. Centralized systems ran inside a small number of trusted zones, and once a user or application cleared the perimeter, everything past it operated on broad implicit trust. That assumption does not survive contact with a modern data lake house. What used to be one system is now a federation of decoupled services, query engines, metadata catalogs, orchestration frameworks, ingestion pipelines, business intelligence tools, and cloud-native storage, scattered across hybrid and multi-cloud infrastructure and held together by APIs rather than a shared network segment. Zero-trust architecture answers this shift with one governing rule: nothing is trusted by default, regardless of where it originates, and every request gets authenticated, authorized, and continuously checked against identity, context, and policy [1]. Lakehouse platforms, which unify data-warehouse-style governance with data-lake-style storage flexibility, are a leading example of this shift already playing out in practice [13].

Picture a representative deployment to see why this issue is not an abstract concern. Analysts query through Looker or Metabase, and engineers connect directly with DataGrip; both paths submit work to Apache Trino, which traces its architectural lineage to Facebook's Presto engine [11], running on Amazon EKS. Trino resolves schema and table metadata through a catalog before reading Apache Iceberg tables sitting in Amazon S3. That is five components, five distinct trust boundaries, and a single compromised service account or misconfigured catalog entry that can expose data across every one of them simultaneously. Zero trust here is closer to a structural necessity than a hardening option, since there is no perimeter left standing to harden.

The cost of getting this wrong is not hypothetical. The global average cost of a data breach reached $4.44 million in 2025, down from $4.88 million the year before, though the average United States breach cost climbed to a record $10.22 million in the same period [17], and 89 percent of enterprises now

Independent Researcher, USA
ORCID: 0009-0009-8847-2267

operate multi-cloud environments, up from 87 percent a year earlier [18]. The scattered, multi-cloud topology described above is not a corner case that only a few organizations need to worry about; it is the condition under which most analytics platforms already operate.

The literature already supplies strong general principles. Kindervag's original argument was that network location should carry zero security weight and that all traffic deserves suspicion until it proves otherwise [2]; Google's BeyondCorp program then showed at real production scale that internal applications could move onto the open internet once every request carried verifiable device and user context instead of leaning on network placement [3]. NIST later distilled these ideas into vendor-neutral tenets in Special Publication 800-207: verify explicitly, apply least privilege, and assume breach [1] and extended them to containerized, API-driven cloud-native workloads specifically in SP 800-207A [5]. Yet recent comprehensive surveys of published zero-trust work find the bulk of it still concentrated on network- and workload-level enforcement, Kubernetes segmentation, service mesh identity, and container isolation, leaving the internal control points of a disaggregated analytical query stack largely untouched [14, 20].

That untouched space is where this paper sits. A single analytical query submitted through a BI tool crosses at least four trust boundaries on its way to an answer: the identity provider authenticating the caller, the query engine planning and executing the request, the metadata catalog resolving which tables and columns exist and who may see them, and the storage layer holding the underlying bytes. Each component enforces its access model, mostly blind to what the others decide. Query engines see execution plans, not always fine-grained sensitivity labels. Storage layers see bucket paths, not column semantics. The metadata catalog is the outlier; it holds a complete map of ownership, schema, classification, and lineage across every path that touches the data, which raises the question this paper argues rather than assumes: should fine-grained authorization live at the catalog instead of being scattered piecemeal across the engine and storage layers around it? Put that plainly, the claim is disputable. An architect chasing minimal latency and fewer moving parts has a reasonable case for engine-native role-based access control instead, and that disagreement is worth taking seriously rather than waving away.

Four contributions follow from taking the catalog-anchored position seriously. The paper derives a reference architecture mapping NIST's zero-trust tenets onto the actual control points of a Trino-and-Iceberg lakehouse stack, identity federation through Okta, OAuth2, and OpenID Connect at the entry point, and a catalog-integrated authorization service (Lakekeeper paired with OpenFGA) as the anchor for access decisions. It formalizes what continuous verification costs in latency and shows, algebraically, how caching claws most of that cost back. It gives the trust-decay idea, a session that was fine at query start but should not be trusted by query end, a composite behavioral risk score model built from signals security teams already watch. And it holds the catalog-anchored design up against engine-native and storage-native alternatives on granularity, consistency, and auditability, stating openly where it loses the comparison and not only where it wins.

2. Method

2.1 Deriving the reference architecture

Building this architecture meant mapping the tenets of NIST SP 800-207 [1] and its cloud-native extension in SP 800-207A [5] onto the actual chain of components in a disaggregated lake house: BI and query-authoring tools up front, a federated identity provider, a distributed SQL engine, a metadata catalog wired to an authorization service, and object storage holding data in an open table format. Authentication and authorization are not treated as properties any one component owns in isolation; they belong to the request path as a whole, in keeping with the zero-trust premise that trust gets re-earned at every hop rather than assumed once and carried forward [4]. Five checkpoints emerged from this mapping rather than a smaller or larger number, because five is the count of distinct trust domains the representative deployment actually crosses; collapsing any two of them into a single check would mean trusting one component to vouch for a domain it does not control, which is the exact assumption zero trust rejects. Figure 1 lays out the resulting trust-boundary structure: five explicit checkpoints between a BI tool's query and the Iceberg data it eventually reaches in S3, read top to bottom from client request to storage retrieval.

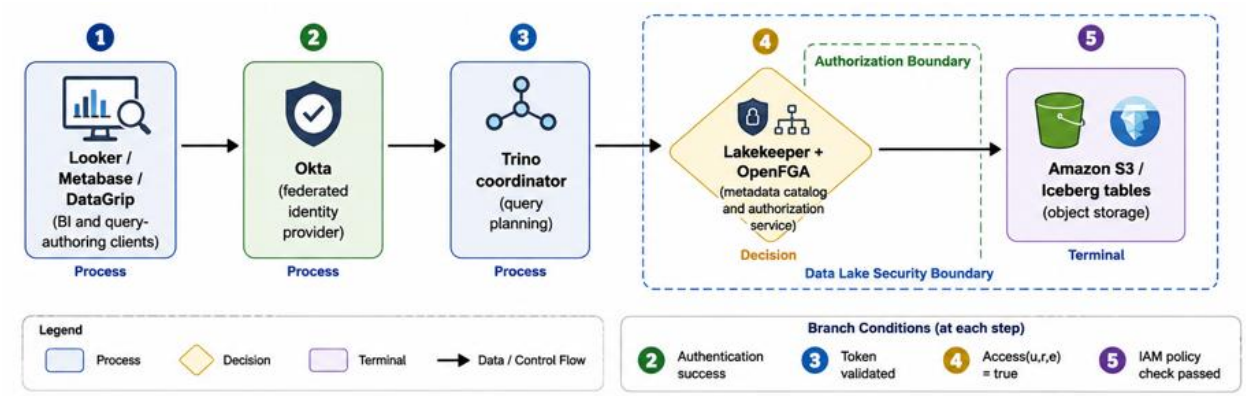

Figure 1: Reference Architecture and Trust-Boundary Structure [1, 5]

2.2 Identity federation and workload authentication

Nothing downstream works if identity is not solid at the front door, so a centralized identity provider, Okta, in the deployment modeled here, authenticates every human and service actor using OAuth2 [7] and OpenID Connect [8] as the federation layer. BI tools authenticate users through Okta-issued OAuth2 flows before a session with the query engine ever opens; DataGrip and similar tools follow the same path, which means no analytics application stores its own long-lived credential. Short-lived, signed access tokens replace stored passwords and propagate across services instead, a meaningful distinction, since a stolen short-lived token exposes a narrow window while a stolen stored credential exposes an open-ended one. Every boundary along the path, from the query coordinator to its workers to the metadata layer beyond, validates an incoming token on its own rather than trusting an upstream check to vouch for it, echoing the older principle that a principal must cryptographically bind to its claimed identity before acting on it [4]; TLS 1.3 [9] protects the tokens themselves in transit and trims the handshake exposure earlier TLS versions carried.

Two extensions push the envelope further. Adaptive authentication folds context, device posture, location, IP reputation, and behavior pattern directly into the authentication decision, so a managed device on a known network clears with minimal friction, while the same account from an unfamiliar device or new geography triggers a step-up check or a temporary restriction. Machine identities get matching treatment: ingestion jobs, orchestration tasks, and service-to-service calls inside the EKS cluster authenticate through workload identity with ephemeral, auto-rotated credentials, closing off one of the more common routes to long-lived credential exposure. Table 1 outlines the identity and token-lifecycle controls applied by this layer, along with the specific risks each control is intended to mitigate.

Control	Mechanism	Risk Narrowed
Federated authentication	Okta-issued OAuth2 / OpenID Connect flows for BI and query-authoring tools [7, 8]	Stored, long-lived credentials inside analytics applications
Short-lived access tokens	Cryptographically signed tokens validated independently at every service boundary [4]	Extended exposure window from a single stolen credential
Transport encryption	TLS 1.3 protecting tokens and payloads in transit [9]	Interception or downgrade of authentication material
Adaptive/risk-based authentication	Device posture, geolocation, IP reputation, and behavioral signals folded into the authentication decision	Authentication from unmanaged devices or anomalous locations
Workload identity for machine actors	Ephemeral, auto-rotated credentials for ingestion, orchestration, and service-to-service calls on EKS	Static API keys embedded in configuration or code

Table 1: Identity Federation and Token Lifecycle Controls

2.3 Catalog-anchored, context-aware authorization

Authentication settles who is asking. Authorization must determine what can be seen, and a static role assignment is too crude once financial, operational, and customer data are on the same platform. The architecture leans on attribute-based access control instead, weighing each request against identity, group membership, sensitivity classification, query intent, device posture, location, time, and any live risk signal [6]. The distinguishing choice is not the ABAC model itself but where the evaluation happens: policy logic sits outside both the query engine and the storage layer, externalized into OpenFGA and wired directly to Lakekeeper, the catalog governing the underlying Iceberg tables. The instinct behind this mirrors Google's Zanzibar system, which proved that authorization holds up better at scale when evaluation is centralized as its own service rather than duplicated inside every application that needs a yes-or-no answer, a claim backed by Zanzibar's own production numbers: sub-10-millisecond 95th-percentile authorization latency at millions of requests per second, with better than 99.999% availability sustained over three years [10]. Zanzibar governs consumer services at an enormous scale; this architecture applies the same centralizing logic to a single organization's analytical estate instead.

In practice, a request from Looker or DataGrip causes Trino to resolve target tables against Lakekeeper, which checks with OpenFGA before returning metadata or data. This approach ensures that one central evaluation is made, rather than several inconsistent ones scattered across every tool that can access the catalog. That placement buys finer control than a table-level allow or deny: hiding sensitive columns outright, filtering rows to a user's business unit, masking part of a field without denying the whole record, and enforcing data residency at the authorization layer rather than through a bolted-on compliance step afterward. Formally, a request clears only when every attribute-level predicate the policy defines comes back true:

$$Access(u,r,e) = P1(u,r,e)\ AND\ P2(u,r,e)\ AND\ \ldots AND\ Pn(u,r,e)$$

with u being the requesting identity, r the resource (table, column, or row), e the environmental context, and each Pi a single predicate, a classification check, a residency check, and a time-window check. Under this structure, adding a governance requirement, means adding a predicate, not rebuilding the authorization system, which is what keeps the model extensible as those requirements shift.

Catalog-level enforcement alone is not enough, though. A query planner, an API gateway, or the storage layer can provide an attacker with a way to bypass the entire model if they honor a request without consulting the catalog, regardless of how carefully the model was designed. Enforcement remains layered: the catalog serves as the primary decision point, while the gateway, planner, and storage access layer also check policy, based on the assumption that preventing bypass is more valuable than maintaining architectural tidiness.

2.4 Decoupling policy from execution

The architecture keeps authentication, authorization, policy evaluation, and audit logging as their own services rather than folding them into the query engine itself. Okta handles authentication; OpenFGA, wired to Lakekeeper, handles authorization; Trino gets to do what it does best, plan and execute distributed queries, while every access decision gets delegated outward to a service built specifically for that job. Amazon EKS supplies the orchestration layer that lets this decoupled set of services scale independently: the query engine's worker pool can grow to meet analytical demand without the authorization service needing to scale in lockstep, and the reverse holds too. This separation also keeps the architecture extensible in practice; adding a new analytics tool, engine, or storage format means integrating it with the existing identity and authorization services rather than reimplementing access control in whatever is added next, which aligns with the open, interoperable posture that cloud-native security guidance generally recommends [12]. The same decoupling also limits the blast radius during an incident: because the query engine holds no authorization logic of its own, compromising Trino does not automatically hand an attacker the authorization service's decisions, and compromising the authorization service does not directly expose the query engine's internal state, which keeps a single component failure from cascading into a full-platform failure.

3. Results and Discussion

3.1 Mapping zero-trust tenets to lake house control points

Table 2 aligns each tenet of NIST SP 800-207 [1] with the component responsible for enforcing it and the mechanism that enforces it. This same five-stage

mapping tracks closely with the identity, network, application/workload, and data pillars CISA's Zero Trust Maturity Model uses to gauge federal and enterprise maturity, which is a reasonable indication that the architecture's checkpoints cover every pillar the model tracks rather than leaning on just one [16]. What the mapping exposes is that one query does not cross one verification point, it crosses at least five in sequence, matching the five steps in Figure 2: authentication at the API layer, authorization against the catalog, planning across coordinators, parallel execution across workers, and retrieval of encrypted data from storage. Any one of those five is a place where a stale token or an expired permission slips through if verification stops being continuous.

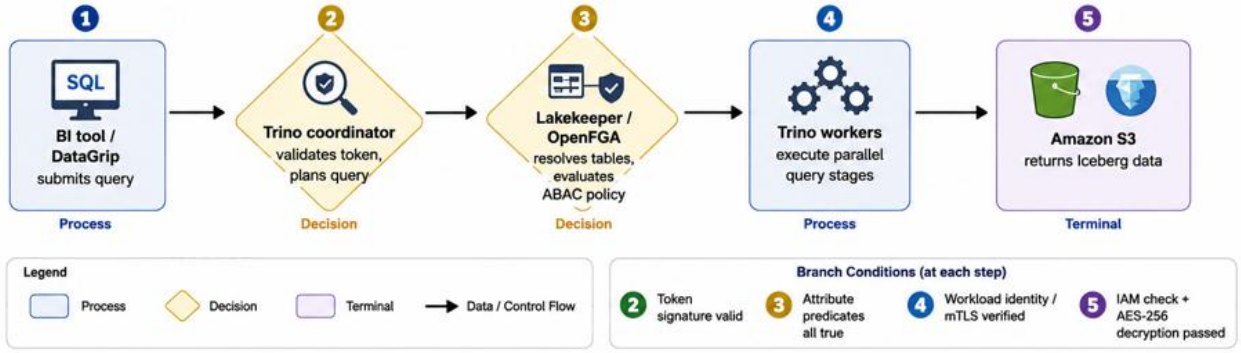

Figure 2: Verification Checkpoints for a Single Analytical Query [1]

Zero-Trust Tenet	Lake House Component	Enforcement Mechanism
Verify explicitly	Identity provider (Okta)	OAuth2 / OIDC token issuance and per-hop validation [7, 8]
Least-privilege access	Metadata catalog (Lakekeeper)	Attribute-based, column/row-level authorization via OpenFGA [6, 10]
Assume breach	Query engine coordinator/workers (Trino)	Independent token validation at every coordinator/worker boundary; mutual TLS
Make policy visible and measurable	Policy control plane (OpenFGA)	Centralized, externalized policy evaluation, auditable independently of the engine
Generate evidence continuously	Storage layer (Iceberg / S3)	Access logging, AES-256 encryption at rest, signed access requests

Table 2: Zero-trust tenets mapped to lake house control points and enforcement mechanisms

Continuous verification is what keeps that five-stage path honest beyond its starting point. OAuth2 tokens issued by Okta must remain valid, and their claims must remain applicable as they move through the Trino coordinator, workers, and Lakekeeper. Mutual TLS and workload identity reinforce trust at each boundary independently, rather than relying on one upstream check to cover everything downstream. This matters most in long-running queries, where conditions can shift mid-flight: an account disabled while a query is still executing, a permission pulled after a policy change, a session a behavioral detector just flagged, all of these should be able to kill an in-flight request rather than let a decision made minutes ago keep governing access to data right now.

That scenario, legitimate at the start and suspect before the finish, is what the trust-decay concept names, and it can be given a shape rather than left as a vague alarm condition. A composite risk score pulls together the behavioral signals a security team is likely watching anyway:

$$R(t) = w1 * x1(t) + w2 * x2(t) + w3 * x3(t) + w4 * x4(t)$$

where x1 through x4 track export volume against baseline, query frequency against baseline, deviation from typical access pattern, and geographic or network anomaly, and w1 through w4 are weights that an organization sets to reflect how much each signal should count. Crossing a defined threshold on R(t) mid-execution becomes a trigger for re-verification or termination rather than just another log line, which turns trust decay from a concept into something the system actually acts on. The weights and threshold themselves are deployment-specific and are not offered here as fixed values; the contribution is the structure the model gives to an otherwise fuzzy idea, not a claim about numbers pulled from a live system.

3.2 The latency cost of continuous verification, and why caching offsets it

None of this comes free. Every extra authentication or authorization check adds latency, and in a high-throughput analytics environment that overhead stacks up across every hop a request passes through. Total latency for a verified query breaks down into base execution time plus the verification cost paid at each hop:

$$T\_total = T\_exec + Sum\ over\ i = 1..n\ of\ (T\_auth, i + T\_authz, i)$$

with T_exec the query's execution time with no verification overhead, and T_auth,i / T_authz,i the authentication and authorization cost at hop i across n hops. Left alone, that sum scales linearly with how many verification points the architecture adds, which is precisely why some architects resist fine-grained, multi-hop authorization in the first place. Caching is the counterweight: when a fraction ci of requests at hop i can be answered from a still-valid cached decision instead of a fresh check, effective latency at that hop drops to

$$T\_eff, i = T\_cache, i * ci + T\_auth, i * (1 - ci)$$

so as the hit rate ci climbs toward one, that hop's contribution shrinks toward the cost of a cache lookup rather than a full round trip. This equation is just the formal version of what the architecture already leans on in practice: cached token validation, cached policy decisions, indexed authorization lookups, and incremental re-validation that checks only what changed since the last pass. The tradeoff underneath the formula is a governance question as much as an engineering one; choosing a cache expiry window really means choosing how long a revoked permission can stay effectively alive before the next mandatory recheck, and that choice deserves the same scrutiny as the access rules themselves.

3.3 Comparing catalog-anchored, engine-native, and storage-native enforcement

Table 3 sets the catalog-anchored model against its two most plausible rivals: access control built natively into the query engine and access control enforced at the storage layer through bucket and object policy.

Enforcement Mechanism	Granularity	Cross-Engine Consistency	Auditability	Latency
Catalog-anchored (OpenFGA + Lakekeeper)	High (column/row/field level)	High (one decision shared by every tool)	High (centralized decision log)	Moderate (external service call per decision)
Engine-native RBAC (Trino)	Moderate (table/role level)	Low (reimplemented per engine)	Moderate (engine-specific logs)	Low (in-process check)
Storage-native IAM (S3 bucket/object policy)	Low (bucket/object path only)	Low (no visibility into query semantics)	High (mature cloud storage logs)	Low (enforced at the storage API)

Table 3: Enforcement mechanism comparison across catalog-anchored, engine-native, and storage-native models.

Catalog-anchored enforcement through OpenFGA against Lakekeeper wins on cross-engine consistency, as every tool querying through the catalog gets the same answer no matter which engine or client sent the request, and wins on semantic granularity, since the catalog alone sees column classification and lineage across every access path. Engine-native RBAC wins on latency, skipping the external service call, but enforces differently for every engine an organization runs, a liability that shows up the moment a second engine joins the stack. Storage-native IAM wins on auditability, riding on cloud storage's already-mature access logs, but it sits below the semantic layer entirely; an S3 policy can lock down a bucket or object path, but it has no vocabulary for hiding a single column because it has no concept of a column at all.

None of that resolves the argument completely. An organization running one engine against one backend, with no plan to add a second, gets little from externalizing authorization and still pays the latency cost and the extra operational dependency for the privilege. The case for catalog-anchored enforcement gets stronger in direct proportion to how many access paths reach the same data, the more tools, engines, and pipelines that touch a given table, the more a single, consistently enforced decision point is worth, and the weaker it looks to

enforce separately and simply hope those separate enforcements stay aligned.

3.4 Extending zero-trust across ingestion, streaming, and metadata

Query execution is not the only place a lake house has to get trust right, and an architecture that locks down query paths while leaving ingestion open has just relocated the weak point rather than closed it. Ingestion pipelines make an obvious target because they are entry points: a compromised or malicious producer can inject corrupted or unauthorized data before any query-time control ever fires, so producers authenticate with federated or workload identity the same way query-time actors do, with access scoped further by dataset ownership and schema validation rather than left open once an initial check clears. Streaming producers and consumers face the same requirement under harsher constraints: they need continuous authentication and authorization while still achieving low-latency, high-velocity throughput. This is precisely the case for which the caching model in Section 3.2 exists, making it workable rather than prohibitive.

Metadata earns the same scrutiny as query paths and ingestion, arguably more, because a catalog governing schema, ownership, and authorization mappings across the whole environment is a high-value target on its own terms; compromising it does not require touching any single query engine, since its decisions apply everywhere at once. Storage security closes the loop: Iceberg tables in S3 should pair encryption at rest using AES-256 with strict IAM policy, bucket isolation, and signed access requests. Table 4 lays out representative risks across ingestion, metadata, and storage alongside the controls that address them.

Risk	Affected Surface	Mitigating Control	Reference
Malicious or corrupted data injection	Ingestion pipelines	Producer authentication via federated/workload identity; dataset-ownership and schema-validation scoping	Section 3.4
Metadata catalog compromise	Lakekeeper/catalog layer	Catalog treated as a high-value control point; authorization decisions centralized and logged	Section 2.3, 3.4
Storage exposure or misconfiguration	Amazon S3/Iceberg tables	AES-256 encryption at rest, strict IAM policy, bucket isolation, signed access requests	Section 3.4
Credential theft and lateral movement	Query coordinator/worker, service-to-service calls	Ephemeral workload identity, mutual TLS, independent per-hop token validation	Section 2.2, 3.1

Table 4: Illustrative risk-to-control mapping across ingestion, metadata, and storage surfaces.

Interoperability is the question this section raises without fully closing. A typical lake house mixes open-source engines, managed cloud services, and internally built tooling, and without standardized identity propagation, token validation, policy formats, audit schemas, and encryption standards across all of it, fragmented enforcement quietly rebuilds the inconsistency the catalog-anchored model exists to remove. Open protocols matter here for exactly this reason; OAuth2, OpenID Connect, and Kubernetes workload identity let a new component join without inventing its own trust model, in line with the interoperability case recent multi-tenant hybrid cloud identity work has made [15].

3.5 Scalability and observability at production scale

An architecture that only holds up at a small scale is not much of an answer for an enterprise lake house, given these platforms routinely serve thousands of users against petabytes of continuously changing data. The decoupling from Section 2.4 is what lets this one scale horizontally: because authentication, authorization, and audit logging run as independent services rather than embedded logic, Kubernetes-native scaling on EKS can grow the query engine's worker pool without forcing a parallel redesign of the identity or policy layers. Observability ties the rest of the architecture together, since none of the preceding controls mean much without visibility into whether they are actually working, logging authentication attempts, authorization decisions, and anomalous access patterns feeds security information and event management platforms, supporting incident investigation and the kind of ongoing compliance evidence regulators increasingly expect. Recent empirical work on service mesh resilience in Kubernetes backs the broader point here: identity-aware, policy-driven enforcement can be layered onto a distributed

system without undermining its operational stability, provided the enforcement layer is built for the failure modes of the platform underneath it [19].

3.6 Limitations

The architecture and its two formal models are analytical contributions, not measured output from a production system, and that distinction matters enough to restate plainly. The latency model in 3.2 and the risk-score model in 3.1 describe structure, how latency accumulates, and how signals combine without asserting specific weights, hit rates, or thresholds since no verified deployment data exists to support such numbers and inventing them would misstate what this paper can actually claim. The architecture also assumes a metadata catalog capable of hosting or integrating an externalized authorization service; a catalog that cannot support these assumptions would need a different anchor point, most plausibly the engine itself, with the consistency costs Section 3.3 already lays out. The comparison in Table 3, finally, is structural analysis rather than benchmarked measurement, and anyone evaluating this architecture for real adoption should test its latency and consistency claims against their workloads before committing to it.

4. Conclusion

Disaggregation made the lake house possible and also makes conventional access control fall short. Once a query's path runs through an identity provider, a query engine, a metadata catalog, and cloud storage in sequence, no single component can hold consistent access control on its own, because no single component sees the whole path. Anchoring fine-grained authorization at the metadata catalog closes that gap; it is where semantic visibility into tables, columns, and lineage is most complete, while identity federation and workload identity handle authentication at the boundaries around that decision point. This discussion is not an argument that catalog-anchored enforcement wins everywhere; Section 3.3 says plainly where engine-native and storage-native alternatives hold up well, and the case made here is strongest exactly where multiple engines and tools converge on the same governed data, a condition that is becoming the norm rather than the exception. The formal models for verification latency and behavioral trust decay extend zero-trust literature that has so far stayed concentrated on network and workload segmentation into the specific territory of disaggregated analytical query engines, territory the existing survey work has not yet mapped. As lakehouse platforms keep adding engines, tools, and machine-driven pipelines to an already federated environment, a single, catalog-anchored authorization layer looks like the more durable choice, not because it erases the tradeoffs discussed here, but because it is the one point built to see the whole environment rather than a handful of disconnected pieces.

Acknowledgments

None.

Author Contributions

Sole author: conceptualization, architecture design, analysis, and writing.

Conflicts of Interest

None declared.

References

[1] Rose, Scott, Oliver Borchert, Stu Mitchell, and Sean Connelly. "Zero trust architecture." NIST special publication 800, no. 207 (2020): 1-52. https://doi.org/10.6028/NIST.SP.800-207

[2] Kindervag, John, S. Balaouras, K. Mak, and J. Blackborow. "No more chewy centers: The zero trust model of information security." Forrester Research 23 (2016). https://crystaltechnologies.com/wp-content/uploads/2017/12/forrester-zero-trust-model-information-security.pdf

[3] Ward, R. and Beyer, B., 2014. Beyondcorp: A new approach to enterprise security. USENIX ;login:, 39(6), pp.6-11. https://www.usenix.org/system/files/login/articles/login_dec14_02_ward.pdf

[4] Lampson, Butler, Martin Abadi, Michael Burrows, and Edward Wobber. "Authentication in distributed systems: Theory and practice." ACM Transactions on Computer Systems (TOCS) 10, no. 4 (1992): 265-310. https://dl.acm.org/doi/abs/10.1145/138873.138874

[5] Chandramouli, Ramaswamy, and Zack Butcher. A zero trust architecture model for access control in cloud-native applications in multi-cloud environments. No. NIST Special Publication (SP) 800-207A. National Institute of Standards and Technology, 2023. https://csrc.nist.gov/pubs/sp/800/207/a/final

[6] Hu, Vincent C., David Ferraiolo, Rick Kuhn, Arthur R. Friedman, Alan J. Lang, Margaret M.

Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. "Guide to attribute based access control (abac) definition and considerations (draft)." NIST special publication 800, no. 162 (2013): 1-54. https://book.ole12138.cn/Book/Guide%20to%20attribute%20based%20access%20control%20%28abac%29%20definition%20and%20considerations.pdf

[7] Hardt, Dick. The OAuth 2.0 authorization framework. No. rfc6749. 2012. Available: https://www.rfc-editor.org/info/rfc6749/

[8] Sakimura, Nat, John Bradley, Mike Jones, Breno De Medeiros, and Chuck Mortimore. "OpenID Connect Core 1.0 incorporating errata set 1." The OpenID Foundation, specification 335 (2014). https://openid.net/specs/openid-connect-core-1_0.html

[9] Rescorla, Eric. The transport layer security (TLS) protocol version 1.3. No. rfc8446. 2018. https://www.rfc-editor.org/info/rfc8446/

[10] Pang, Ruoming, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski et al. "Zanzibar: Google's Consistent, Global Authorization System." In 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 33-46. 2019. https://www.usenix.org/conference/atc19/presentation/pang

[11] Sethi, Raghav, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian James Sun, Nezih Yigitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. "Presto: SQL on Everything." In Proc. IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 2019, pp. 1802-1813. https://ieeexplore.ieee.org/abstract/document/8731547

[12] Cloud Native Computing Foundation, "Cloud Native Security Whitepaper," Version 2, CNCF, San Francisco, CA, USA, 2022. https://www.cncf.io/wp-content/uploads/2022/06/CNCF_cloud-native-security-whitepaper-May2022-v2.pdf

[13] Armbrust, Michael, Ali Ghodsi, Reynold Xin, and Matei Zaharia. "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics." In Proceedings of CIDR, vol. 8, no. 1, p. 28. 2021. https://15721.courses.cs.cmu.edu/spring2023/papers/02-modern/armbrust-cidr21.pdf

[14] Syed, Naeem Firdous, Syed W. Shah, Arash Shaghaghi, Adnan Anwar, Zubair Baig, and Robin Doss. "Zero trust architecture (zta): A comprehensive survey." IEEE access 10 (2022): 57143-57179. https://ieeexplore.ieee.org/abstract/document/9773102

[15] Deochake, Saurabh, and Vrushali Channapattan. "Identity and access management framework for multi-tenant resources in hybrid cloud computing." In Proceedings of the 17th International Conference on Availability, Reliability and Security, pp. 1-8. 2022. https://dl.acm.org/doi/abs/10.1145/3538969.3544896

[16] Cybersecurity and Infrastructure Security Agency, "Zero Trust Maturity Model," Version 2.0, CISA, Washington, DC, USA, 2023. https://www.cisa.gov/sites/default/files/2023-04/CISA_Zero_Trust_Maturity_Model_Version_2_508c.pdf

[17] IBM. Cost of a Data Breach Report 2025. 2025. https://www.ibm.com/reports/data-breach

[18] Luxner, Tanner. "Cloud computing trends: Flexera 2024 State of the Cloud Report," Flexera, 2024. https://www.flexera.com/blog/finops/cloud-computing-trends-flexera-2024-state-of-the-cloud-report/

[19] Singh, Shubham, Cristina Hava Muntean, and Shaguna Gupta. "Resilient microservices: an investigation into Istio effectiveness in Kubernetes." Cluster Computing 29, no. 1 (2026): 27. https://link.springer.com/article/10.1007/s10586-025-05750-x

[20] Weinberg, Abraham Itzhak, and Kelly Cohen. "Zero Trust Implementation in the Emerging Technologies Era: Survey." arXiv e-prints (2024): arXiv-2401. https://ui.adsabs.harvard.edu/abs/2024arXiv240109575I/abstract