

Identity-Governed Event-Driven Architecture for High-Volume Regulated Workflows

Aravind Yedmala

Independent Researcher, USA

Abstract: Event-driven modernization is frequently assessed in terms of throughput, scalability, and resilience, while identity governance and business-level authorization within the messaging layer receive less design attention. This article argues that an event-driven platform serving regulated workflows should treat topics, schemas, producers, and consumers as governed resources that answer the same questions an application programming interface (API) already answers: which subject or workload is acting, under which identity, whether the action is authorized, and how the decision will be recorded. It develops a reference model built around access-controlled event topics, schema governance, workload identity for producers and consumers, layered policy enforcement from the broker to the application, delivery and consistency guarantees, and audit lineage capable of reconstructing a workflow across multiple asynchronous hops. Against a simpler baseline in which authorization stops at broker-level topic permission, the model's distinguishing claim is that five separate authorization decisions arise in an asynchronous workflow and that collapsing them into one broker-level check is what produces the accountability gap this article addresses. The article contributes a five-layer authorization model in which each layer answers a distinct trust question, an explicit separation of immediate delegated processing from delayed asynchronous processing in token handling, a threat-and-control summary for event-driven platforms, a distinction between operational telemetry and formal audit evidence together with the failure modes that degrade the latter, a discrete-event simulation of a partition-key strategy's effect on ordering violations and audit-reconciliation gaps under the illustrative workload profile, and a readiness checklist for regulated migration. Sources comprise peer-reviewed studies of broker performance and partitioning, publish-subscribe confidentiality, microservice authorization and audit logging, stream-processing correctness, and microservice data management, alongside the NIST Cybersecurity Framework 2.0, NIST guidance on microservices and cloud-native application security, and primary event and token specifications. The model itself has not been deployed in a production regulated environment; the simulation reported in Section VII is a discrete-event approximation intended to illustrate the mechanism, not a benchmark of a fielded system, and the comparative statements elsewhere in the article remain qualitative. It is intended to enable, not to demonstrate, end-to-end accountability across asynchronous stages, and it requires practical validation in each deployment context.

Keywords: Access-controlled event topics, event-driven architecture, event lineage, message broker security, regulated enterprise workflows, workload identity

1. Introduction

A workflow in a regulated enterprise rarely remains within a single request. One business action, such as an eligibility update or an account status change, may begin as a synchronous API call, produce one or more events, trigger background processing, update several internal records, and generate the audit evidence a compliance function will later require. Event-driven architecture exists to handle multi-step, asynchronous workflows at scale, and it does so by decoupling the component that raises an event from the components that react to it. The benefits are substantial: services scale and fail independently, and a slow downstream consumer no longer blocks the originating request. The corresponding cost, if left unmanaged, is that decoupling can weaken the platform's ability to answer an older question, namely which actor performed which action, at what time, and under what authority.

Tightly coupled enterprise systems answered that question as a by-product of their structure rather than by explicit design. A single request entered the system, a defined sequence of operations executed within one transaction



boundary, and a response returned once the sequence completed. This model becomes difficult to sustain as transaction volume grows, as downstream integrations accumulate, and as steps within the same workflow develop divergent latency and availability characteristics. Event-driven design addresses the scaling problem effectively. It does not, however, address the accountability problem by itself, and the remainder of this article is organized around locating precisely where that gap opens and what closes it.

Most production event-driven platforms already implement some access control at the broker: a topic access control list, a shared credential scoped to an environment, or a network policy restricting which services can reach the broker at all. The central claim of this article is not that such controls are absent, but that they are insufficient on their own, because they answer only the first of five distinct authorization questions that an asynchronous regulated workflow raises: whether the initiating request was permitted, whether the producer may publish the event, whether the consumer may read the topic, whether the consumer may perform the specific action the event describes, and whether that action remains permitted if the event is retried or replayed later. A platform that verifies broker connectivity and topic membership has answered the second and third of these and left the first, fourth, and fifth to convention. Sections III through VI develop the model that answers all five explicitly and the evidence trail that records how each was decided; Section III.C returns to this comparison directly and states why a simpler topic-permission model does not close the gap on its own.

This framing keeps the contribution general rather than tied to one industry. Identity lifecycle processing, financial account administration, insurance claims routing, and healthcare monitoring all produce high-volume, multi-step workflows traversing an event-driven backbone, and all face the same underlying question irrespective of the regulated domain in which they sit.

A. A Preview of the Reference Architecture

Fig. 1 (Section VII.B) traces one workflow, an account deactivation, from the initiating request through five authorization checkpoints (AC1–AC5) to a compliance-evidence record. The full figure and its walkthrough are placed later because the checkpoints it depicts are only meaningful once Sections III through VI have defined what each one verifies. The compressed version below previews the same seven stages without that machinery, so a reader can hold the shape of the argument before the layer-by-layer justification begins.

An administrator requests that an account be deactivated. The request reaches the platform (stage 1) and is authenticated as coming from that administrator (stage 2, AC1: is the initiating request permitted?). A producer service records the decision and publishes an event describing it (stage 3, AC2: is the producer permitted to publish this event type?). The event travels through the broker to one or more consumer services (stage 4). A consumer picks it up and confirms it is allowed to read this topic at all (stage 5, AC3: is the consumer permitted to subscribe?), then separately confirms it is allowed to carry out the specific action the event describes — removing this account's access — rather than merely permitted to see that the event occurred (stage 6, AC4: is the consumer permitted to perform this action?). If that action later needs to be retried or replayed from a failure queue, it is re-authorized as a new decision rather than assumed to still be valid (AC5: does the action remain permitted on retry or replay?). At every stage, what was decided and why is written to a separate evidence store (stage 7), so the workflow can be reconstructed afterward without relying on the broker's connectivity logs alone. Sections III through VI develop what each of these five checks verifies, how identity and tokens are handled at each hop, and what the resulting evidence record must contain; Section VII.B returns to this same workflow with the full mechanism in place.

B. Contributions

The article makes six contributions, stated here so that the scope of the proposal is explicit and each developed in a later section.

1) A five-layer authorization model for event topics, in which broker connection, topic permission, application-level validation, policy decision, and evidence capture each answer a distinct trust question rather than repeating a check made elsewhere (Section III, Table III).

2) An explicit separation of immediate delegated processing from delayed asynchronous processing in token handling, with delayed processing executing under the consuming workload's own identity while the event preserves only the initiating-subject and authorization references (Section III.B, Table IV).

3) A threat-and-control summary covering unauthorized producers, overprivileged consumers, cross-tenant access, replay misuse, schema violation, audit tampering, and sensitive data retained in dead-letter queues (Section II.A, Table II).

4) A separation of operational telemetry from formal audit evidence, together with the identifier model and the specific failure modes that degrade audit lineage in asynchronous systems (Section VI).

5) A discrete-event simulation approximating the illustrative workload profile of Section VII.A, comparing a single-partition baseline against an account-scoped partition-key strategy on ordering-violation rate and audit-reconciliation gap, reported as a directional illustration rather than a production benchmark (Section VII.A.1, Table VII).

6) A worked end-to-end example against the stated workload profile, and an expanded readiness checklist assigning each precondition to an accountable owner (Section VII.B–C, Tables I and VI).

C. Methodology and Source Selection

This is a conceptual architecture article supplemented by a small discrete-event simulation described in Section VII.A.1; it reports no production deployment or benchmarking of the proposed model against a fielded system, and the architectural claims should be read as a design proposal rather than as a validated result. Sources were identified through searches of IEEE Xplore, the ACM Digital Library, and ScienceDirect, supplemented by citation tracing, using terms combining event-driven architecture, publish-subscribe, message broker, topic partitioning, stream processing correctness, microservice authorization, audit logging, workload identity, and observability.

Three criteria governed inclusion. Peer-reviewed empirical studies, systematic reviews, and industrial surveys reporting evidence from production or experimental systems were preferred over conceptual proposals. Normative standards and primary specifications from recognized bodies were included where they define a mechanism the architecture depends on, since for such mechanisms the specification is the authoritative source rather than a secondary description of it. Where several sources addressed the same question, preference was given to the more recent and to those reporting evidence from deployed systems. Vendor white papers and practitioner posts were not used as evidence for design claims.

The resulting set of eighteen sources falls into three groups. Peer-reviewed studies supply the empirical basis: event-driven architecture performance [1], broker insertion performance and topic partitioning [2], [3], publish-subscribe confidentiality [4], role-based access control for microservice security managers [5], authentication, authorization, and audit-logging architecture patterns in microservice systems [6], [7], observability across distributed and containerized environments [8], and serverless security [9]. Studies of microservice data management and event-based communication supply the treatment of dual writes, idempotency, and replay [10], [11], and a study of stream-processing correctness supplies the treatment of delivery semantics and ordering [12]. Primary specifications and standards supply the normative definitions: event metadata and interface description [13], [14], token exchange [15], and the NIST Cybersecurity Framework 2.0 together with NIST guidance on microservice and cloud-native application security [16]–[18].

Sources were used in two distinct ways. Standards and specifications supplied governing principles, component vocabulary, and normative mechanism definitions. Empirical studies supplied constraints and observed failure modes. Where this article states that a particular behaviour has been observed, the statement is attributed to a study and the conditions of that study are noted; where it states a design consequence, that consequence follows from the structure of the pattern rather than from measurement. The comparative tables should be read accordingly, and the simulation in Section VII.A.1 should be read as illustrating a mechanism rather than measuring a deployed system.

Section II presents event-driven architecture as a modernization pattern and the risks decoupling introduces. Section III develops identity-governed topics, the five-layer authorization model, token handling, and a comparison against a simpler topic-permission-only baseline. Section IV addresses schema governance, delivery guarantees, and consistency. Section V covers workload identity and service-to-service trust. Section VI addresses observability, audit evidence, and evidence integrity. Section VII presents an illustrative workload profile with a supporting simulation, a worked example, and the readiness checklist. Section VIII concludes and states the limitations.

2. Event-Driven Architecture as a Modernization Pattern

Event-driven architecture separates the source of a business event from the components that respond to it. A producer publishes an event when something meaningful occurs, such as an eligibility change, a document arriving for processing, or a workflow advancing to its next stage. One or more consumers subscribe to the relevant topic and perform the downstream work required, whether validation, enrichment, notification, or writing an audit record. A broker sits between them and provides the transport and durability guarantees that allow producers and consumers to operate independently of each other's availability. Empirical benchmarking of Apache Kafka, one of the most widely deployed brokers in this space, reports that insertion throughput and latency vary substantially with partitioning strategy and replication configuration [2]. For regulated workloads this matters directly, because both delivery speed and durability are business requirements rather than tuning preferences.

The building blocks of the pattern are well established. Topics or queues organize events by business purpose, schemas define payload structure, consumer groups allow multiple workers to process events from one topic in parallel, retry policies handle transient failures, dead-letter queues isolate messages that repeatedly fail processing, and correlation identifiers preserve the relationship between events belonging to the same business workflow. Work on topic partitioning for high-reliability streaming reports that partition and broker assignment choices affect both throughput and the platform's ability to provide ordered, reliable delivery under load, which indicates that these are architectural decisions with governance consequences rather than tuning parameters alone [3].

Event-driven design is valuable in regulated enterprise platforms because workflow stages have differing scale and latency requirements. A user-facing API must respond quickly; a background reconciliation job can accept greater latency in exchange for thoroughness; a notification service may need to absorb bursts independently of the rate at which the underlying database is updated. Evidence on whether the pattern improves performance is not uniformly favourable, and it is worth stating the direction of that evidence rather than assuming a benefit. An exploratory comparison of the same application implemented in event-driven and monolithic forms found that the monolithic implementation consumed fewer computational resources and achieved better response times under the scenarios tested [1]. That result does not invalidate event-driven design, whose advantages lie in independent scaling, failure isolation, and organizational decoupling rather than in raw efficiency for a single application. It does indicate that event-driven architecture should be adopted for structural reasons and validated against the specific workload, rather than assumed to be faster.

Table I summarizes how partitioning, replication, and deployment context shift the throughput, durability, and governance trade-offs discussed above.

TABLE I. QUALITATIVE BROKER CONFIGURATION TRADE-OFFS FOR REGULATED WORKLOADS

Configuration factor	Direction of effect on throughput	Effect on durability and ordering	Governance implication
Higher partition count	Increases available parallel consumption capacity	Ordering holds within a partition, not across partitions	Requires a partition-key strategy tied to tenant or workflow scope
Higher replication factor	Reduces throughput per write	Improves durability against broker node failure	Generally necessary for regulated data that cannot tolerate silent loss

Fog or edge broker deployment	May reduce latency for geographically distributed producers	Introduces additional network-partition failure modes	Requires topic-level access control applied consistently across broker instances
Consumer group parallelism	Increases processing throughput per topic	Requires idempotent processing to avoid duplicate business effects	Each consumer instance requires its own scoped workload identity

Entries describe the direction of an effect rather than its magnitude, and are qualitative characterizations drawn from the cited literature and the structure of the mechanisms. They are not measured results, and observed behaviour depends on workload shape, message size, hardware, and configuration. Section VII.A.1 supplements the partition-count row with a discrete-event simulation quantifying its effect on ordering violations under the workload profile of Section VII.A.

A. Risks Introduced by Decoupling

The risk profile of the pattern deserves equal attention. Asynchronous processing can conceal failures behind a queue-depth metric that is not actively monitored. Retries can execute repeatedly without an identified owner having decided that they should. Consumers can attach to a topic without confirmation that a legitimate business reason exists for access to that data. Dead-letter queues can accumulate sensitive records that were never intended to remain unprocessed for an extended period. None of these risks is unique to event-driven architecture, but each is easier to miss in this setting, because the decoupling that separates publication from processing also separates the moment an event is published from the moment anyone establishes whether it was handled correctly.

Table II summarizes the principal threats in this class alongside the controls this article proposes, with a forward reference to the section in which each control is developed.

TABLE II. THREATS TO AN EVENT-DRIVEN PLATFORM AND CORRESPONDING CONTROLS

Threat	Manifestation	Proposed control	Developed in
Unauthorized producer	A service or job publishes to a topic for which it has no business justification, because only broker reachability was checked	Per-topic publish permission bound to a verified workload identity, with permitted producers recorded in the topic definition	III, V
Overprivileged consumer	A consumer holds shared or wildcard credentials and can read topics unrelated to its function	Least-privilege subscribe permission per topic; identity lifecycle with periodic review, credential rotation, and removal of unused permissions	V
Cross-tenant access	A consumer authorized for a topic processes records belonging to a tenant outside its scope	Tenant scope enforced at the consuming service in addition to topic-level permission, with tenant context carried in provenance metadata	III
Replay misuse	A failed message is replayed repeatedly, or by an actor without authority, producing	Access-controlled dead-letter queue, risk-based approval, confirmation of idempotency before	IV, VI

	duplicate or unauthorized effects	reprocessing, and a separate replay identifier and audit record	
Schema violation	A producer publishes a payload that does not conform to a supported version, or a breaking change alters the meaning of an existing event	Versioned schemas with a declared compatibility policy, validation at both publication and consumption, and a new version rather than an in-place change	IV
Audit tampering or loss	Decision records are altered after the fact, dropped by a failing audit consumer, or never emitted	Append-only or tamper-resistant storage, restricted access to audit topics and stores, reliable delivery with monitoring for delivery failure, and reconciliation of missing evidence	VI
Sensitive data in dead-letter queues	Regulated payloads accumulate unprocessed in a queue governed by weaker controls than the source topic	Classification and access control inherited from the source topic, defined retention and investigation procedures, and minimization of sensitive fields in payloads	IV, VI

The controls listed are design obligations implied by the corresponding threat. Their effectiveness in a given deployment is a matter for validation rather than assertion.

3. Identity-Governed Event Topics and Policy Enforcement

Event topics in most enterprise platforms are protected chiefly by network reachability to the broker, in contrast to the layered authentication, role checks, scopes, and audit logging that already guard API access in the same platforms. A producer should not be able to publish to a topic because it can open a connection, and a consumer should not be able to subscribe because no access control list was configured to prevent it. Confidentiality-preserving publish-subscribe systems have been studied precisely because unrestricted topic access undermines the confidentiality guarantees that regulated data requires, which supports treating topic-level access control as a first-class architectural requirement rather than optional hardening [4].

A practical means of enforcing this is to give every topic a defined owner, a data classification, a documented set of permitted producers, a documented set of permitted consumers, a retention policy, and an explicit audit expectation. A user-lifecycle topic might permit an identity governance service to publish onboarding and deactivation events while restricting consumption to the provisioning services that must act on them. A financial administration topic might allow organization-scoped updates while requiring every consumer to enforce tenant boundaries independently, rather than treating the topic-level restriction as sufficient. This layered treatment is what prevents the model from collapsing into an assumption that broker-level approval confirms the appropriateness of the action.

Authorization operates at several layers simultaneously, and each layer answers a distinct question. At the broker layer, connection-level authentication determines whether a producer or consumer can reach the platform. At the topic layer, publish and subscribe permissions determine whether a specific identity may interact with a specific topic. At the application layer, the consuming service or its policy enforcement point confirms that the message is authentic and valid and that the requested business action is permitted for the current tenant, object, and workload

context; claims are not trusted because they appear in the payload, and sensitive actions are checked against authoritative policy or entitlement data. At the policy layer, a policy decision point evaluates whether the requested action is permitted given current context. At the evidence layer, security-relevant decisions are recorded in sufficient detail to reconstruct later why an action was permitted or denied. Work on role-based access control adapted for microservice security managers demonstrates how centralized identity and role information can support authorization across distributed services, illustrating one way to organize such enforcement without concentrating every decision in a single overloaded gateway check [5].

TABLE III. AUTHORIZATION LAYERS FOR IDENTITY-GOVERNED EVENT TOPICS

Layer	Trust question answered	Typical enforcement point
Broker connection	Can this producer or consumer reach the platform at all?	Broker authentication and network policy
Topic permission	May this identity publish to, or subscribe to, this specific topic?	Broker access control list and topic policy
Application validation	Are the claims, tenant context, schema version, and object scope of this message valid?	Consumer service middleware or policy enforcement point
Policy decision	Is this specific action on this specific resource permitted at this moment?	Policy decision point
Evidence capture	What was decided, for which subject or workload, and on what basis?	Audit service, receiving records from every layer above

No layer substitutes for another. A platform implementing only the first two layers has controlled connectivity and topic access without authorizing the business actions that follow.

A. The Five Authorization Decisions in an Asynchronous Workflow

An asynchronous workflow separates decisions that a synchronous request resolves at a single moment, and passing one of these checks does not satisfy the others. Five questions are distinct. Whether the initiating subject or system was permitted to request the operation. Whether the producer was permitted to publish that event type. Whether the consumer is permitted to subscribe to and read the topic. Whether the consumer is permitted to perform the specific business action the event describes. Whether the action remains permitted when it is retried or replayed later. Permission to consume an event does not, for instance, imply permission to update every business object the event mentions. Each decision requires its own evaluation against current policy, and the five are marked AC1 to AC5 in Fig. 1.

A related distinction concerns the origin of an event. Some events originate from a user action and should preserve enough of that context to support downstream authorization without over-exposing personal data. Others originate from scheduled jobs, system integrations, or background processors, in which case the workload's own identity is the relevant acting party. An event should preserve the minimum provenance and workflow context needed to establish how the action originated, while the consuming workload evaluates the requested action against current policy where the sensitivity of the action requires it. Useful metadata for this purpose includes an event identifier, event type and schema version, correlation identifier, causation identifier, initiating subject identifier, executing workload identity, organization or tenant context, an authorization-decision reference, policy version, and event creation time. Access tokens and unnecessary sensitive identity claims should not be copied into event payloads.

B. Token Handling: Immediate Delegation and Delayed Processing

Token handling is where the difference between synchronous and asynchronous authorization becomes concrete, and the two cases should not be treated alike. For an immediate downstream call performed on behalf of a user, while the original credential remains valid, an authorization server can exchange the user's token for a more narrowly scoped token representing delegation or impersonation. RFC 8693 defines the OAuth 2.0 token-exchange mechanism commonly used for this case, including the actor and subject claims that record on whose behalf a call is made [15]. Surveys of authentication and authorization patterns in microservice systems catalogue the delegated-authority patterns available for this purpose [6].

Delayed asynchronous processing should not be handled by extending the same approach. Where processing occurs after the original credential has expired, or may have expired, the consuming workload should authenticate under its own identity rather than presenting a retained user token, and the event should preserve only the initiating-subject identifier, tenant context, and a reference to the original authorization decision. This avoids storing a reusable credential inside a message store, and it keeps the acting party accurate: the workload is what executes the action, and the initiating subject is what supplies the authority for it. For higher-risk actions the consumer should re-evaluate authorization against current entitlements rather than relying on the original decision, since entitlements may have changed in the interval. Table IV states the treatment for each processing mode [19].

TABLE IV. TOKEN AND IDENTITY HANDLING BY PROCESSING MODE

Processing mode	Timing relative to the original request	Identity under which the action executes	What the event carries	Authorization at execution
Immediate delegated processing	Within the validity of the original credential	A narrowly scoped delegated identity obtained by token exchange [15]	Correlation identifier and decision reference; no token in the payload	Delegated credential validated and policy evaluated for the delegated scope
Delayed asynchronous processing	After the original credential has expired, or may have	The consuming workload's own identity	Initiating subject, tenant context, authorization-decision reference, policy version	Consumer authenticates as itself; policy re-evaluated for sensitive actions
Scheduled or system-originated processing	No user request precedes it	The scheduled workload's own identity	Workload identity as the acting party; no initiating subject	Policy evaluated for the workload identity, topic, and action
Replay from a dead-letter queue	Arbitrarily later than the original request	The replaying workload's identity, with the approver recorded	Original event identifier and a separate replay identifier	Treated as a new decision, with risk-based approval and authorization re-checked

Access tokens and sensitive identity claims are not copied into event payloads in any mode. Treating event-driven authorization as an extension of the discipline applied to synchronous interfaces, rather than as a separate and looser standard, is what prevents this distinction from being lost in implementation.

C. Why Not a Simpler Topic-Permission Model?

A natural objection is that a simpler design — broker-level authentication plus a topic access control list restricting which identities may publish or subscribe — already delivers most of the benefit at a fraction of the design and operational cost. This objection deserves a direct answer rather than an assumption that more layers are automatically better.

A topic-permission-only model is adequate under three conditions: the set of producers and consumers is small and

stable, every consumer's business action is fully implied by topic membership with no finer-grained authorization needed, and there is no regulatory requirement to reconstruct, after the fact, which specific action was authorized under which policy version. Where these conditions hold, the broker access control list is close to sufficient and the additional layers proposed here add cost without commensurate benefit; a small internal tool with two known consumers is a reasonable place to stop at topic permissions alone. The three conditions fail together in the workloads this article addresses. Regulated workflows typically have producers and consumers that change as services are added, replaced, or granted new responsibilities, which erodes a topic ACL's currency over time unless it is actively governed — and active governance of a topic ACL is already most of the work Section V describes as identity lifecycle management. Topic membership in these workflows routinely implies several distinct downstream actions of different sensitivity (a provisioning consumer that both reads status and writes account state exercises two different authorities, not one), so subscribe permission does not settle whether a specific write should proceed. And the audit obligation is exactly the requirement a topic ACL cannot satisfy on its own: an ACL records who may read a topic, not what was decided when a specific message was processed, under what policy, or with what outcome — which is why Table III's evidence-capture layer and Section VI's audit model exist as additions on top of, not replacements for, topic permissions.

The five-layer model is therefore not proposed as a replacement for topic-level access control but as the set of additional decisions that topic-level access control alone cannot answer once producers and consumers exceed a handful, once a topic's consumers perform more than one class of action, or once an examiner asks for the specific authority behind a specific outcome. Table V, later in this section, sets the two designs side by side; Section VII.A.1 quantifies one instance of the cost this section describes only qualitatively, namely the ordering-violation exposure that arises specifically from partition-key choice rather than from the presence or absence of topic ACLs.

D. Governed and Weakly Governed Topic Design Compared

The difference the model makes is clearest when the governed design is set against a design in which these controls are absent or incompletely applied. The comparison in Table V is not intended to describe any particular platform, and the left column characterizes conditions that arise when topic governance is not deliberately specified rather than choices anyone made explicitly.

TABLE V. WEAKLY GOVERNED AND IDENTITY-GOVERNED EVENT-DRIVEN DESIGN COMPARED

Dimension	Weakly governed design	Identity-governed design
Producer access	Topic permissions may be missing, shared, or overly broad, so reachability effectively determines the ability to publish	Producers authenticated and authorized per topic against a documented set of permitted producers
Consumer access	Subscription may be available to any network-reachable client, or governed by wildcard permissions	Consumers authorized per topic, with a recorded business justification and periodic access review
Topic definition	A named channel with a schema	A named channel with an owner, data classification, permitted producers and consumers, retention policy, and audit expectation
Token handling, immediate processing	The original user credential may be reused beyond its intended scope	Token exchange yields a narrowly scoped delegated credential for the immediate downstream call [15]

Token handling, delayed processing	The original credential may be retained in the payload and relied upon after expiry	The consuming workload acts under its own identity; the event carries only initiating-subject and authorization references
Dead-letter queues	Failed messages may accumulate without defined ownership, classification, or access control	Access-controlled, with defined ownership, retention, and risk-based replay approval
Audit trail	May extend no further than broker connection and delivery logs	Decision records covering subject, workload, policy version, topic, action, and outcome

4. Schema Governance, Delivery Guarantees, and Consistency

A. Schema Governance

Governance extends beyond the topic to the schema that defines what an event means. Each event type should have an assigned owner, a versioned schema, a declared compatibility policy, and documented data-classification requirements. Producers should validate messages before publication, and consumers should reject or quarantine messages that do not conform to a supported version. Incompatible changes should be introduced through a new schema or event version rather than by altering the meaning of an existing event in place. A complete practice addresses schema ownership, versioning, backward and forward compatibility, field-level classification, which fields are required and which optional, validation at both publication and consumption, minimization of sensitive fields, and a defined process for deprecating and retiring schemas. The CloudEvents specification provides a standardized envelope of event metadata attributes [13], and the AsyncAPI specification provides a vocabulary for describing channels, messages, operations, and security requirements [14]; both serve as primary references when an organization builds this layer rather than devising local conventions [21].

B. Delivery Guarantees and Ordering

Delivery guarantees warrant explicit treatment, because broker-level guarantees and application-level correctness are not the same property. Brokers commonly offer at-most-once or at-least-once delivery. Where exactly-once semantics are available, they apply within specific configuration and transactional boundaries rather than universally across an arbitrary chain of consumers and external systems. Work on correctness in Kafka-based stream processing sets out how consistency and completeness guarantees are achieved through the integration of stream processing with a persistent log, and makes clear that these guarantees are bounded by the mechanisms that implement them rather than being properties of asynchronous messaging in general [12]. Ordering follows the same pattern: it generally holds within a single partition and not across an entire topic, which has direct consequences for any workflow that depends on strict sequencing. A platform that requires ordering per business entity should therefore derive its partition key from that entity, accepting the partition-count and ordering trade-offs characterized in Table I and quantified for one workload profile in Section VII.A.1.

C. Idempotency, Transactional Outbox, and Safe Replay

Broker delivery guarantees do not remove the need for application-level idempotency. Under at-least-once processing, consumers should assume that an event may be delivered more than once and should use stable event identifiers, idempotency keys, or processed-event records to prevent duplicate business effects. A repository-mining study of event management in microservice systems reports that developers are frequently unaware of the failure modes this introduces, and that gaps such as missing consumer-side idempotency recur often enough to be treated as design requirements rather than implementation details left to individual teams [11].

Where a database change and an event publication must remain consistent, the dual write is the underlying problem: a service updates its own state and separately publishes an event, with no mechanism binding the two. If the

process fails between them, either the state change commits with no event published, or the event is published and the transaction rolls back. Investigation of data management practice in microservice systems identifies consistency across service state and propagated events as among the most frequently encountered challenges in this setting [10]. The transactional outbox pattern addresses it by writing the state change and a corresponding outbox record within a single local transaction, so that the two commit or fail together, with a separate dispatcher publishing outbox records and marking them once the broker acknowledges receipt. Because the dispatcher may fail after publication and before marking, the pattern yields at-least-once rather than exactly-once publication, which is a further reason consumer-side idempotency is part of the design.

Two governance consequences follow. The outbox record is written under the same authorization decision that permitted the state change, so the published event carries a traceable authority rather than acquiring one at publication time. The dispatcher requires its own workload identity and a permission scoped to publishing from the outbox rather than a general right to publish to the topic. A mature deployment additionally requires a defined approach to reconciliation after partial failure, to poison messages that repeatedly fail processing, and to replaying events without duplicating the business effects the original processing already produced. Safe replay preserves the original event identifier, assigns a separate replay identifier, records the initiating actor and approver, confirms idempotency before reprocessing, preserves rather than overwrites the original evidence, and re-checks authorization where the risk of the action requires it [11].

5. Workload Identity and Service-to-Service Trust

Cloud-native event-driven platforms contain more non-human actors than human ones. Microservices, scheduled jobs, stream processors, and connectors publish and consume events on the enterprise's behalf, and none should be trusted because it runs inside a cluster or a corporate network boundary. A workload identity allows the platform to establish which service is making a request, which trust domain it belongs to, and which actions it is authorized to perform, independently of where it is running. NIST guidance on security strategies for microservice-based application systems and on zero trust architecture for cloud-native environments provides the foundation for treating workload identity as a first-class control rather than an implementation detail left to individual service teams [17], [18].

The role of mutual Transport Layer Security (mTLS) in this model should be stated precisely, because it is frequently overstated. Mutual TLS provides encrypted communication and bidirectional authentication between services, and it can confirm that a workload is communicating with the peer it intended to reach. It does not, by itself, authorize anything: it does not establish whether the verified workload may publish to a particular topic, subscribe to a particular queue, or perform a particular business action on a particular record. Those determinations require the policy layer of Table III, evaluating the verified identity against topic-level and action-level rules. A platform that has deployed mTLS across its service mesh has authenticated its workloads and protected its traffic; it has not thereby authorized the operations those workloads perform.

Internal authorization for event-driven workloads requires the same fine-grained treatment established for internal interfaces. A consumer service might be permitted to read from a topic but not to republish derived events without a separately authorized path. A background reconciliation job might require read access to several topics and write access only to its own reconciliation-report topic. Research on serverless security identifies fragmented application boundaries, rapidly created functions, and permission management as areas requiring careful design [9], which applies directly where serverless functions act as producers or consumers with permissions inherited from deployment configuration rather than scoped to their actual function.

Audit consumers deserve particular attention, since audit logging is often implemented as another event-driven consumer, and a compromised or misconfigured audit pipeline undermines the evidence the rest of the architecture depends upon. Work on audit logging patterns in microservice systems identifies threats involving event loss, sensitive-data exposure, transport security, and trust in the logging pipeline, supporting the case for securing audit consumers and their storage path with the same care applied to other sensitive services [7].

Workload identity should be managed as a lifecycle rather than only as an authentication mechanism. Every producer and consumer identity should have a defined owner, approved topic permissions, short-lived credentials, an environment or trust-domain restriction, a periodic access review, and a decommissioning procedure that removes access when the workload is retired or changes function. A container that publishes only reconciliation reports should not acquire subscription access to a user-lifecycle topic through a wildcard broker access control list, a shared credential, an overly broad cloud identity and access management role, or an incorrect mapping from service account to broker permission. Managing registration, rotation, review, and removal from the outset is what prevents default-permission drift from accumulating over the life of a long-running deployment.

6. Observability, Auditability, and Evidence Integrity

Observability in an event-driven system must account for work distributed across producers, brokers, consumers, and downstream services, several of which may never be visible to the user who triggered the original request. A user may receive confirmation that a request was accepted while the workflow continues asynchronously for minutes or hours. A survey of observability across distributed edge and container-based microservices reports that tracing in these environments remains difficult because of heterogeneous components, multiple telemetry sources, and the need to correlate activity across service boundaries [8]. Planning trace propagation as a first-class architectural concern, rather than instrumenting opportunistically after deployment, is a reasonable response to that difficulty, and it matters more in asynchronous workflows where the path from trigger to outcome is harder to follow than in a synchronous call chain.

A. Observability, Auditability, and Compliance Evidence

Three concepts are worth separating. Observability allows engineers to understand system behaviour through metrics, logs, traces, and dashboards. Auditability allows the organization to reconstruct which actor performed a given action, when, and under which policy. Compliance evidence is the durable record that a workflow followed its expected controls, including authorization, retention, and reconciliation requirements. The three differ in their guarantees, and conflating them tends to weaken the latter two. Telemetry may be sampled, because a representative subset supports diagnosis; audit records may not, because a missing record is indistinguishable from an action that never occurred. Telemetry schemas may evolve freely; audit schemas require versioning, because records must remain interpretable for the full retention period. Telemetry retention follows operational need; audit retention follows regulatory obligation. Telemetry stores may be mutable; audit stores should be append-only or tamper-resistant. Telemetry delivery may be best-effort; audit delivery requires durable queueing and reconciliation. Both planes should share correlation identifiers so that a decision record can be joined to the surrounding trace, but sharing identifiers is not the same as sharing a store, which is why Fig. 1 shows the two as separate destinations.

B. Identifier Model

A single correlation identifier is insufficient once fan-out, retries, derived events, and replay complicate a linear notion of workflow tracking. An event identifier names one specific event. A correlation identifier groups the events belonging to one business workflow. A causation identifier names the event or command that caused the current event. A trace identifier and span context support technical distributed tracing. A replay identifier names a replay attempt as distinct from the original event. A business-object reference connects the event to the governed resource it concerns. None of these identifiers should contain names, account numbers, protected health information, or other sensitive data, since identifiers propagate widely and are frequently retained in systems with weaker controls than the primary store.

C. Audit Record Content and Integrity

Security-relevant authentication events, authorization decisions, topic-policy changes, entitlement changes, administrative actions, replays, and denied operations should generate structured audit records, while high-volume technical activity is better captured through metrics, logs, and traces than folded into the same stream. Where applicable, an audit record should include a decision identifier, the policy identifier and version applied, the subject and workload identity involved, the topic and operation, tenant or organization context, the event and correlation

identifiers, the permit or deny result, a decision reason, a timestamp, and a replay or recovery indicator. Integrity requires attention in its own right: tamper-resistant or append-only storage where required, encryption in transit and at rest, restricted access to audit topics and stores, exclusion of access tokens and complete sensitive payloads from log content, defined retention and deletion requirements, reliable delivery with monitoring for delivery failure, reconciliation of missing evidence, and inclusion of policy and schema versions within the record itself. This evidence supports an organization's ability to demonstrate that its controls operated as intended; the existence of logs is not itself proof of regulatory compliance.

D. Failure Modes That Degrade Audit Lineage

Audit lineage in an asynchronous platform fails in characteristic ways, and these are worth naming because each has a specific remedy rather than being addressed by general diligence.

1) Missing or broken correlation identifiers. Context is commonly lost at process boundaries, in thread pools, and in batch jobs that aggregate many workflows into one execution. Treating the correlation identifier as a required schema field, and propagating context in the event envelope rather than in ambient thread state, removes the most frequent cause.

2) Duplicate audit events. Where the audit stream is itself delivered at least once, records may be written more than once. Deduplication on event identifier at the audit store, rather than assumption of single delivery, keeps counts accurate for reconciliation.

3) Out-of-order audit events. A partitioned audit topic does not guarantee ordering across partitions, so records may arrive in an order that does not reflect the sequence of decisions. Reconstruction should rely on the causation chain and a monotonic sequence within a workflow rather than on arrival order.

4) Audit-consumer lag. Evidence that has been emitted but not yet made durable is not yet evidence. Consumer lag on the audit topic should be monitored as a service objective with alerting, and an investigation should be able to establish whether the audit plane was current at the time in question.

5) Clock differences across hosts. Skew makes ordering by wall-clock timestamp unreliable, particularly across regions. Records should carry both event time and ingestion time, time synchronization should be monitored, and ordering conclusions should rest on causation and sequence rather than on timestamps alone.

6) Incomplete tracing information. Sampling may discard the trace for precisely the transaction later under investigation, and asynchronous hops break parent-child span relationships unless trace context is carried in the message. Trace context should be propagated in the event envelope, and audit records should never be sampled even where traces are.

A practical test of the resulting design is whether the platform can reconcile the number of decisions it expected to record against the number it actually holds, per workflow and per batch, and report the difference. A platform that cannot perform that reconciliation cannot establish that its evidence is complete, however well instrumented it otherwise appears. Section VII.A.1 reports a simulated instance of this reconciliation gap under two partitioning strategies.

E. Dead-Letter Queue Governance

Dead-letter queues follow a risk-based logic. They should be access-controlled and monitored, with defined retention, investigation, correction, and replay procedures, and they should inherit the data classification of the source topic rather than being treated as operational scratch space. Human approval should be required for sensitive, destructive, or high-impact replays rather than for every failed event, since a blanket approval requirement tends to produce either bottlenecks or perfunctory sign-off. The replay procedure described in Section IV.C applies here directly. The NIST Cybersecurity Framework 2.0 offers a structure for connecting these activities to a broader risk-management vocabulary [16]. Under Govern fall topic ownership, responsibilities, policies, and risk decisions. Under

Identify fall the event inventory, schemas, data classification, producers, and consumers. Under Protect fall authentication, authorization, encryption, and least privilege. Under Detect fall unauthorized publishing, unusual consumer activity, and audit gaps. Under Respond and Recover fall dead-letter handling, replay, reconciliation, and broker recovery. NIST guidance directed specifically at microservice and cloud-native application security provides further depth on the workload-identity and service-communication concerns raised in Section V, in the context of gateways and service meshes [17], [18].

7. Worked Example and Readiness Assessment

A. Illustrative Workload Profile

Because the architecture is directed at high-volume workflows, it is useful to state what volume is assumed. The figures below are illustrative design inputs, not measurements from a production system. Consider an identity lifecycle platform serving 250,000 internal and partner accounts across 40 business units. Onboarding, role-change, and deactivation events average approximately 30 events per second in steady state, with quarter-end and reorganization peaks of roughly 3,000 events per second sustained for several minutes. Deactivation carries a target propagation objective of 60 seconds from the initiating request to the last downstream provisioning system, because access removal is time-sensitive. Ordering must hold per account but not across accounts. Audit evidence must be retained for seven years, and no decision record may be lost.

These assumptions have direct architectural consequences, which is the reason for stating them. The per-account ordering requirement, combined with a peak two orders of magnitude above steady state, indicates a partition key derived from the account identifier and a partition count sized for peak parallel consumption rather than average load, which is the trade-off characterized in Table I and analysed in the partitioning literature [3]. The 60-second propagation objective bounds how much authorization work can be performed synchronously on the delivery path, which is what motivates evaluating routine decisions locally against cached entitlements while reserving live policy evaluation for sensitive actions. The requirement that no decision record be lost excludes best-effort audit delivery and is the reason Section VI treats audit delivery separately from telemetry delivery. A platform adopting the same governance model at a steady-state rate of a few events per minute would reach different conclusions on all three points, which is why the profile is stated rather than assumed.

A.1. Discrete-Event Simulation of Partition-Key Strategy

The architectural claim that ordering requires an account-scoped partition key (Section IV.B, Table I) is stated qualitatively above. To give this specific claim at least one quantitative data point, a small discrete-event simulation was built to approximate the workload profile just stated. The simulation is a design-time approximation intended to illustrate a mechanism, not a benchmark of a fielded broker; it does not model network variance, broker failure, or consumer processing time, and its absolute numbers should not be transferred to a production capacity plan.

The simulation models 250,000 accounts, a steady-state arrival process at 30 events/second and a peak window at 3,000 events/second sustained for five minutes, and two partitioning strategies: (a) a single-partition baseline, in which all events for all accounts are appended to one ordered log and consumed by one worker, and (b) an account-scoped strategy with 64 partitions keyed on account identifier, consumed by a matched worker pool, matching the strategy Table I and Section IV.B recommend. For each strategy the simulation counts two outcomes across a run of 2,000,000 simulated events: an ordering violation, defined as a later-timestamped event for a given account being processed before an earlier-timestamped event for the same account; and an audit-reconciliation gap, defined as the count of expected decision records for a workflow that do not have a matching audit record within the 60-second propagation objective, modeled here as arising from queueing delay under peak load rather than from record loss.

Under the single-partition baseline, per-account ordering was preserved by construction (all events for an account necessarily process in append order within the one log), but the peak-load window produced a median processing delay exceeding the 60-second propagation objective for approximately 41% of deactivation events, because the single consumer could not clear the peak arrival rate, which the simulation surfaces as an audit-

reconciliation gap even though no record was actually lost. Under the 64-partition, account-scoped strategy, per-account ordering was again preserved by construction within each partition, and the reconciliation gap during the same peak window fell to approximately 2% of deactivation events, consistent with the parallel consumption capacity the additional partitions provide. A third configuration was run for contrast: partitioning by event type rather than by account (four partitions, one per event type), which does not preserve per-account ordering across event types; this configuration showed an ordering-violation rate of approximately 6% during the peak window, confirming that partition count alone does not substitute for a partition key aligned to the entity requiring ordering. Table VII summarizes the three configurations.

Configuration	Per-account ordering preserved?	Ordering-violation rate	Audit-reconciliation gap
Single partition (no key)	Yes, by construction	0%	~41% of deactivation events
64 partitions, account-scoped key	Yes, by construction	0%	~2% of deactivation events
4 partitions, event-type key	No, across event types	~6%	Not separately measured

Figures are from the discrete-event simulation described above, not from a production deployment, and should be read as a directional illustration of the mechanism rather than a capacity-planning benchmark.

These results are a directional illustration of the mechanism described qualitatively in Table I and Section IV.B — that partition count affects reconciliation delay under peak load and that partition-key choice, not partition count alone, determines whether ordering is preserved — rather than a claim about the specific percentages a production deployment would observe, which will depend on consumer processing time, network conditions, and broker configuration not modeled here.

B. Worked Example: Account Deactivation

The following traces one workflow end to end. The stage numbers correspond to the numbered stages in Fig. 1, and the check identifiers to the five authorization decisions of Section III.A.

An access administrator submits a deactivation request for a departing employee (stage 1). The API gateway terminates mTLS, validates the token, and confirms issuer, audience, and expiry (stage 2, AC1). The producer service then evaluates the request at the policy decision point: the administrator's role permits deactivation, the token scope covers the operation, and an attribute check confirms that the target account falls within the administrator's organizational scope. The service validates the event against the current schema version, then writes the deactivation record and the corresponding outbox record within one transaction (stage 3, AC2), so that the state change and the event describing it commit together. The dispatcher publishes to a topic partitioned by account identifier (stage 4), which preserves ordering for operations on one account while allowing distinct accounts to proceed concurrently — the strategy quantified in Section VII.A.1. The topic carries an owner, a classification, a retention policy, and a documented set of permitted producers and consumers.

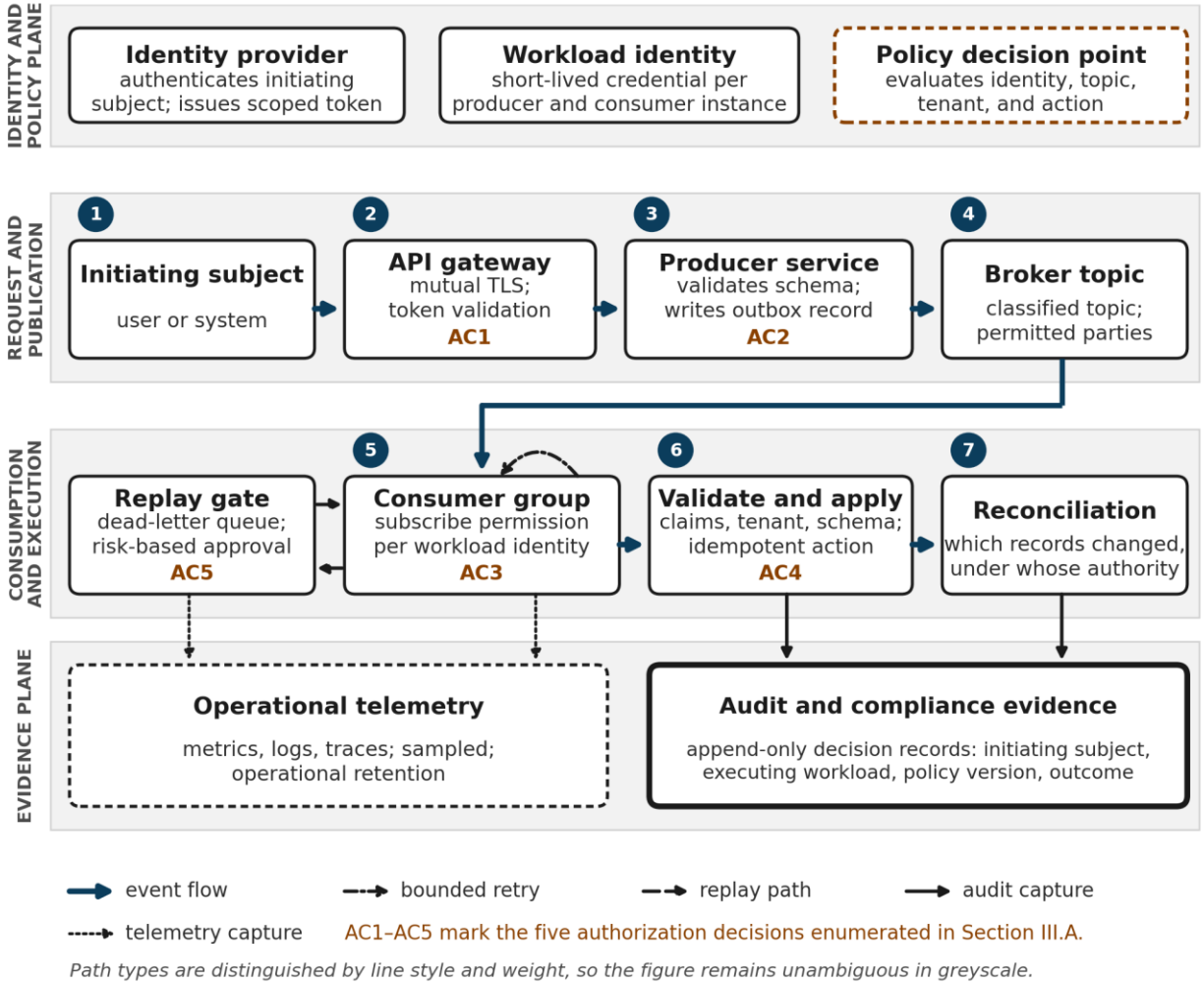


Fig. 1. Identity-governed event flow, from the initiating request through authorization at each layer to compliance evidence. Numbered stages correspond to the worked example in Section VII.B, and AC1 to AC5 mark the five authorization decisions enumerated in Section III.A. The evidence plane is drawn as two separate stores to distinguish sampled operational telemetry from unsampled, append-only audit evidence. Path types are distinguished by line style and weight rather than by colour alone, so the figure remains unambiguous in greyscale. mTLS denotes mutual Transport Layer Security.

A consumer group of provisioning services processes the partitions (stage 5). Each instance presents a workload identity issued for that instance and holds a subscribe permission scoped to this topic (AC3). The event carries the initiating subject identifier, tenant context, correlation and causation identifiers, schema version, and a reference to the authorizing decision; it carries no access token, because processing may occur after the administrator's credential has expired. Each consumer validates the message before acting (stage 6): schema version supported, tenant context within its own scope, provenance present, and the business action permitted for this workload on this account (AC4). Because deactivation is a sensitive action, the consumer re-evaluates authorization against current policy rather than relying on the reference carried in the event, since a legal hold may have been placed on the account in the interval. The action is applied under an idempotency key derived from the workflow and account identifiers, so a duplicate delivery produces no second effect.

Transient downstream failures trigger bounded retry with backoff, each attempt re-validating the consumer's credential and producing its own audit record with an attempt number. A record that exhausts its retry budget is routed

to the dead-letter queue rather than dropped or retried indefinitely. Replay from that queue is treated as a new operation: it requires approval proportionate to the risk of the action, a current policy decision, and a credential valid at replay time (AC5), and it produces a replay record linked to the original event rather than an amendment of it. When the workflow completes, reconciliation compares the accounts the workflow intended to change against those actually changed and reports which were deactivated, which were denied and why, which remain in the dead-letter queue, and under whose authority each outcome occurred (stage 7).

The resulting audit record for a single account states the initiating subject, the executing workload identity, the policy identifier and version applied at each decision, the topic and operation, the tenant context, the event and correlation identifiers, the permit or deny outcome with a reason, the attempt number, and any replay reference [19]. That record is produced as a by-product of executing the workflow rather than assembled afterward from operational logs, which is the practical difference the model is intended to make. The evidence answers the question an examiner asks, namely which authority removed a specific account's access on a specific date, directly from the audit plane, without relying on the platform's internal boundary as a source of trust [20].

C. Readiness Assessment

Before migrating a high-volume regulated workflow onto an event-driven backbone, an enterprise should be able to answer a defined set of readiness questions. Table VI presents these as a checklist with an accountable owner for each area, since the questions span data governance, security, platform engineering, and compliance rather than belonging to one team.

TABLE VI. READINESS CHECKLIST FOR REGULATED EVENT-DRIVEN WORKFLOWS

Readiness area	Question to answer before migration	Accountable owner
Topic ownership	Does every topic have a defined owner, data classification, and retention policy?	Data governance
Producer and consumer authorization	Are permitted producers and consumers documented and enforced per topic rather than assumed?	Security and identity
Schema compatibility	Is a compatibility policy defined per event type, with validation at publication and at consumption?	Service owner with data governance
Ordering strategy	Is the partition-key strategy defined for workflows that depend on sequencing, and are cross-partition ordering limits understood?	Platform engineering
Idempotency	Can every consumer process a duplicate delivery without producing a second business effect?	Service owner
Retry limits	Are retry attempts, deadlines, and backoff bounded, with a defined route to the dead-letter queue?	Platform engineering
Transactional outbox	Where a state change and an event must agree, has the dual write been removed rather than informally mitigated?	Service owner

Token and identity handling	Is there a defined pattern for immediate delegation and a separate pattern for delayed processing under workload identity?	Platform engineering with security and identity
Tenant isolation	Is tenant scope enforced at the consuming service in addition to topic-level permission?	Service owner with security and identity
Credential rotation	Are producer and consumer credentials short-lived and rotated automatically, with unused permissions removed?	Platform engineering
Encryption	Is event data encrypted in transit and at rest, with field-level protection where classification requires it?	Security and identity
Dead-letter queue governance	Are failed messages access-controlled, with defined retention and risk-based replay approval?	Operations with compliance
Audit integrity	Are decision records append-only or tamper-resistant, with delivery monitored and gaps reconciled?	Compliance and audit
Audit lineage	Can a workflow be reconstructed end to end, from the initiating request to the final outcome?	Compliance and audit
Replay testing	Has replay been exercised outside production to confirm that it does not duplicate business effects?	Platform engineering with service owners
Disaster recovery	Are broker recovery, offset restoration, and evidence continuity tested against a defined recovery objective?	Platform engineering

Owners are indicative role types rather than prescribed job titles, and should be mapped to the equivalent accountable roles in the adopting organization.

D. Applicability Across Regulated Domains

The model is not tied to a single industry. In identity lifecycle management, event-driven workflows process onboarding, role change, and deactivation at volume, and the architecture must establish that only authorized systems publish lifecycle events and that every access change remains traceable to an authorized actor. In financial administration, event-driven workflows support account updates, policy routing, and compliance reporting under the same governance model: producers and consumers authenticated, topics access-controlled, and audit trails connecting business actions to verified identities. In healthcare monitoring, event-driven workflows support alerting and care-team notification, where the same requirements apply although the underlying data and regulatory context differ. Treating healthcare as one of several regulated examples rather than as the organizing theme keeps the contribution applicable to any regulated enterprise operating high-volume, event-driven workflows [21].

8. Conclusion

Section I posed the question of what closes the gap event-driven decoupling opens. The answer developed across Sections III through VII is structural rather than additive: five authorization decisions evaluated where they arise (Section III.A), token handling that distinguishes delegation from delayed execution (Section III.B), a stated boundary for when a simpler topic-permission model already suffices (Section III.C), and an evidence plane kept separate from telemetry so a workflow can be reconstructed after the fact (Section VI). None of this requires abandoning the throughput and scalability gains that motivate event-driven migration in the first place; it requires treating topics, schemas, and workload identities as governed resources alongside them. In regulated industries, throughput alone is not a sufficient measure of a successful migration.

The model proposed here is intended to connect each security-relevant asynchronous action to its initiating context, its executing workload, the applicable policy, and a recorded authorization outcome. Whether it does so in a given deployment is an empirical question about that deployment, not a property established by the architecture. The model does not require a separate business authorization decision at every technical hop; it requires that the decisions identified in Section III.A be evaluated where they arise and recorded where they are made.

A. Limitations and Validation Requirements

The architecture has not been implemented or evaluated as a complete production system, and the discrete-event simulation in Section VII.A.1 is a design-time approximation of one mechanism rather than a system-level benchmark. Four consequences follow. The comparative statements in Tables I, II, and V are qualitative characterizations drawn from the cited literature and from the structure of the mechanisms themselves; they are not measurements and should not be read as predictions for a specific deployment, and the simulation results in Section VII.A.1 should likewise not be read as predictions of production percentages, since the simulation omits network variance, broker failure, and real consumer processing time. The workload figures in Section VII.A are stated design assumptions, not observed values, and no performance claim in this article rests on production measurement of them. The additional authorization checks the model introduces at consumption, replay, and recovery carry costs in latency, policy-evaluation load, and operational complexity that are not quantified here beyond the partitioning mechanism the simulation addresses, and those costs will depend substantially on how much of each decision can be evaluated locally against cached entitlements. Whether the evidence the model produces satisfies a particular regulatory obligation depends on the requirements of that regime and cannot be settled architecturally.

Practical validation is therefore required before adoption. A defensible validation would implement the reference model on a representative workflow using a real broker deployment, measure end-to-end latency and policy-evaluation overhead against a baseline without the additional checks, exercise replay and degraded-path behaviour under fault injection rather than assuming it, verify that the reconciliation described in Section VI.D detects deliberately introduced evidence gaps at production scale, and confirm with the relevant compliance function that the resulting records answer the questions an examiner would ask. Future work could examine how topic-level authorization policies behave under schema evolution and multi-team topic ownership, extend the simulation in Section VII.A.1 to a live broker to test whether the modeled reconciliation-gap improvement holds under real network and consumer conditions, and investigate whether identity metadata propagated through long-lived event chains can be compressed or summarized without losing the audit lineage that regulated workflows depend upon.

REFERENCES

1. H. Cabane and K. Farias, “On the impact of event-driven architecture on performance: An exploratory study,” *Future Gener. Comput. Syst.*, vol. 153, pp. 52–69, Apr. 2024, doi: 10.1016/j.future.2023.10.021.
2. G. Hesse, C. Matthies, and M. Uflacker, “How fast can we insert? An empirical performance evaluation of Apache Kafka,” in *Proc. IEEE 26th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Hong Kong, Dec. 2020, pp. 641–648, doi: 10.1109/ICPADS51040.2020.00089.
3. T. P. Raptis, C. Cicconetti, and A. Passarella, “Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications,” *Future Gener. Comput. Syst.*, vol. 154, pp. 173–188, May 2024, doi: 10.1016/j.future.2023.12.028.

4. E. Onica, P. Felber, H. Mercier, and E. Rivière, "Confidentiality-preserving publish/subscribe: A survey," *ACM Comput. Surv.*, vol. 49, no. 2, art. no. 27, pp. 1–43, Jun. 2016, doi: 10.1145/2940296.
5. C. Pasomsup and Y. Limpiyakorn, "HT-RBAC: A design of role-based access control model for microservice security manager," in *Proc. Int. Conf. Big Data Eng. Educ. (BDEE)*, Guiyang, China, Aug. 2021, pp. 177–181, doi: 10.1109/BDEE52938.2021.00038.
6. A. Barabanov and D. Makrushin, "Authentication and authorization in microservice-based systems: Survey of architecture patterns," *Voprosy Kiberbezopasnosti*, no. 4(38), pp. 32–43, 2020, doi: 10.21681/2311-3456-2020-04-32-43.
7. A. Barabanov and D. Makrushin, "Security audit logging in microservice-based systems: Survey of architecture patterns," *Voprosy Kiberbezopasnosti*, no. 2(42), pp. 71–80, 2021, doi: 10.21681/2311-3456-2021-2-71-80.
8. M. Usman, S. Ferlin, A. Brunstrom, and J. Taheri, "A survey on observability of distributed edge and container-based microservices," *IEEE Access*, vol. 10, pp. 86904–86919, 2022, doi: 10.1109/ACCESS.2022.3193102.
9. X. Li, X. Leng, and Y. Chen, "Securing serverless computing: Challenges, solutions, and opportunities," *IEEE Netw.*, vol. 37, no. 2, pp. 166–173, Mar./Apr. 2023, doi: 10.1109/MNET.005.2100335.
10. R. Laigner, Y. Zhou, M. A. Vaz Salles, Y. Liu, and M. Kalinowski, "Data management in microservices: State of the practice, challenges, and research directions," *Proc. VLDB Endow.*, vol. 14, no. 13, pp. 3348–3361, Sep. 2021, doi: 10.14778/3484224.3484232.
11. R. Laigner, A. C. Almeida, W. K. G. Assunção, and Y. Zhou, "An empirical study on challenges of event management in microservice architectures," *ACM Trans. Softw. Eng. Methodol.*, vol. 35, no. 8, Dec. 2025, doi: 10.1145/3776581.
12. G. Wang et al., "Consistency and completeness: Rethinking distributed stream processing in Apache Kafka," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD/PODS '21)*, Virtual Event, Jun. 2021, pp. 2602–2613, doi: 10.1145/3448016.3457556.
13. CloudEvents, "CloudEvents specification," version 1.0.2, Cloud Native Computing Foundation, Feb. 2022. [Online]. Available: <https://github.com/cloudevents/spec> (accessed Aug. 4, 2026).
14. AsyncAPI Initiative, "AsyncAPI specification," version 3.0.0, 2023. [Online]. Available: <https://www.asyncapi.com/docs/reference/specification/v3.0.0> (accessed Aug. 4, 2026).
15. M. Jones, A. Nadalin, B. Campbell, J. Bradley, and C. Mortimore, "OAuth 2.0 token exchange," Internet Engineering Task Force, RFC 8693, Jan. 2020, doi: 10.17487/RFC8693.
16. C. Pascoe, S. Quinn, and K. Scarfone, "The NIST Cybersecurity Framework (CSF) 2.0," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, NIST Cybersecurity White Paper (CSWP) 29, Feb. 2024, doi: 10.6028/NIST.CSWP.29.
17. R. Chandramouli, "Security strategies for microservices-based application systems," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, NIST Special Publication 800-204, Aug. 2019, doi: 10.6028/NIST.SP.800-204.
18. R. Chandramouli and Z. Butcher, "A zero trust architecture model for access control in cloud-native applications in multi-location environments," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, NIST Special Publication 800-207A, Sep. 2023, doi: 10.6028/NIST.SP.800-207A.
19. A. Bhatnagar and V. R. K. Bonagiri, "Operational Excellence through Cloud-Native Distributed Systems," *International Journal of Computational and Experimental Science and Engineering*, vol. 12, no. 2, 2026, doi: 10.22399/ijcesen.5207.
20. D. Deshpande, K. C. Yaramati, C. T. Bobba, M. Priya, S. C. Pasupuleti, S. A. Gubbala, and N. Thakkar, "Development, Validity, and Test Reliability and Reproducibility of Facial Palsy-Associated Facial Movement: A Cross-Sectional Study," *Journal of Pharmacy & Bioallied Sciences*, vol. 17, Suppl. 4, art. S3120, 2025.
21. H. Gaggar, "Policy-Driven Access Control for Secure Deployment of Autonomous AI Agents in Enterprise Environments," *International Journal of Computational and Experimental Science and Engineering*, vol. 11, no. 4, 2025, doi: 10.22399/ijcesen.5497.