NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 8 Issue 4, Pg. No. 65-78, AIJCST-V8I4P106, 2026
https://doi.org/10.63282/3117-5481/AIJCST-V8I4P106

Original Article

Zero-Trust DevSecOps Framework for Event-Driven Microservices across Multi-Cloud Environments

***Nalini Sivakumar**

Independent Researcher, USA.

Abstract:

The rapid adoption of event-driven microservices and multi-cloud computing has transformed enterprise software development by enabling scalable services, asynchronous communication, continuous delivery, and flexible infrastructure deployment. However, these architectural advantages introduce security challenges associated with distributed identities, dynamic workloads, message-broker access, cross-cloud communication, software supply chains, and inconsistent security policies. Traditional perimeter-based security and isolated development-stage security testing are insufficient for environments in which applications, services, and workloads continuously change across heterogeneous cloud platforms. This paper proposes a Zero-Trust DevSecOps Framework for Event-Driven Microservices across Multi-Cloud Environments that integrates identity-centric access control, continuous security validation, event-level protection, software supply-chain assurance, and runtime threat detection throughout the software development lifecycle. The proposed framework combines zero-trust principles with automated security gates, workload identity, policy-as-code, mutual Transport Layer Security (mTLS), event-schema validation, secrets management, container security, and distributed observability. Its architecture is organized into five interconnected layers: secure development and software supply-chain management, identity and policy governance, event-driven communication protection, multi-cloud deployment and orchestration, and continuous runtime monitoring. A design-oriented research methodology is employed to develop the framework and evaluate its conceptual suitability against conventional DevSecOps, perimeter-oriented security, and partially integrated zero-trust approaches. Two comparative tables and two conceptual figures describe the architecture, security controls, evaluation dimensions, and expected operational trade-offs. The discussion indicates that integrating security policies across development, deployment, messaging, and runtime operations can improve security consistency, traceability, and incident containment, although these benefits require empirical validation. The framework contributes a unified architectural approach for organizations seeking to secure asynchronous microservices without unnecessarily compromising deployment agility or distributed-system scalability.

Keywords:

Zero Trust, DevSecOps, Event-Driven Architecture, Microservices Security, Multi-Cloud Computing, Software Supply Chain Security, Policy-as-Code, Workload Identity, Continuous Security Monitoring, Cloud-Native Security.

Article History:

Received: 24.05.2026

Revised: 25.06.2026

Accepted: 04.07.2026

Published: 13.07.2026

Copyright by the Author(s). This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-NC-ND 4.0) License (https://creativecommons.org/licenses/by-sa/4.0/)

1. Introduction

1.1. Background and Motivation

Cloud-native application development has increasingly shifted toward microservices architectures, in which independently deployable services communicate through lightweight APIs, event brokers, and asynchronous messaging systems. This architectural approach allows development teams to release individual capabilities independently, scale selected workloads according to demand, and isolate failures within specific service boundaries. Event-driven architectures extend these advantages by allowing services to publish and consume events without requiring direct, synchronous communication between producers and consumers. Platforms such as Apache Kafka, cloud-managed messaging services, and event-routing systems support this interaction model across distributed enterprise environments. Nevertheless, the distribution of application logic introduces additional security dependencies that cannot be adequately addressed through network perimeter protection alone.

Multi-cloud adoption further increases architectural flexibility by allowing organizations to distribute workloads across different cloud providers according to regulatory requirements, availability objectives, geographical constraints, and operational preferences. At the same time, each provider may implement distinct identity services, network controls, secret-management mechanisms, workload orchestration features, and security-monitoring interfaces. When an event producer operates in one cloud and its consumer operates in another, the organization must establish trust across multiple administrative and technical boundaries. A compromised identity, incorrectly configured message topic, exposed credential, or malicious software dependency can consequently affect services beyond the environment in which the original weakness occurred.

The zero-trust security model provides a relevant foundation for addressing these challenges. Rather than treating network location or organizational ownership as sufficient evidence of trust, zero trust requires explicit authentication and authorization, evaluates contextual information, and limits access to the resources necessary for an approved activity. NIST Special Publication 800-207 establishes foundational zero-trust principles for enterprise architectures, while NIST's Secure Software Development Framework emphasizes integrating security practices throughout software development rather than treating security as a final-stage activity (Rose et al., 2020; Souppaya et al., 2022).

DevSecOps complements these principles by integrating security controls into development workflows, automated testing, infrastructure provisioning, deployment, and operational feedback. However, combining zero trust and DevSecOps does not automatically secure event-driven systems. Conventional pipelines may scan source code and container images while overlooking event-topic permissions, consumer-group access, schema compatibility, replay attacks, cross-cloud identity federation, and authorization changes after deployment. Likewise, runtime monitoring may identify suspicious behavior without connecting it to the source-code revision, dependency set, build provenance, or policy decision that enabled the behavior.

The central motivation for this research is therefore to establish a unified framework in which trust is continuously evaluated across the complete lifecycle of event-driven microservices. Security decisions should remain consistent from code development and artifact construction to event publication, message consumption, workload execution, and incident response. Such integration is particularly important for financial platforms, digital commerce, healthcare applications, industrial monitoring, and other distributed systems in which event integrity, service availability, and traceable authorization are essential.

1.2. Problem Statement

Existing security approaches frequently implement application security, identity management, event-broker authorization, cloud infrastructure protection, and runtime monitoring through separate tools and administrative processes. Although each control may provide meaningful protection, fragmented implementation can create inconsistent policies, duplicated responsibilities, incomplete audit trails, and delays in identifying relationships between development-time weaknesses and production incidents.

These problems become more pronounced when microservices communicate asynchronously across cloud boundaries. A message may be authenticated at the broker but insufficiently validated by its consumer. A container may pass vulnerability scanning but later obtain excessive permissions through a deployment configuration. A service identity may be legitimate within one cloud but inadequately constrained when federated into another. Moreover, retries, event replay, duplicate delivery, and delayed processing can complicate the investigation of security incidents. The research problem is to design an integrated security framework that applies

zero-trust principles throughout DevSecOps while accounting for the distinctive communication patterns and operational requirements of event-driven microservices distributed across multiple cloud providers.

1.3. Research Objectives

The study pursues five objectives:

- To develop a layered security architecture integrating zero trust, DevSecOps automation, and event-driven microservices protection.
- To define consistent identity, authorization, and policy-enforcement mechanisms across heterogeneous cloud environments.
- To incorporate event-specific controls for message integrity, schema validation, replay resistance, and least-privilege broker access.
- To connect software supply-chain assurance and deployment validation with continuous runtime monitoring.
- To establish a conceptual evaluation framework for comparing security coverage, operational overhead, scalability, and governance consistency.

1.4. Research Contributions

The primary contribution is a lifecycle-oriented architectural framework that connects software development security, workload identity, event-broker protection, multi-cloud deployment governance, and runtime response. Unlike approaches that concentrate primarily on pipeline scanning or network access, the proposed design treats the event lifecycle and the software delivery lifecycle as interdependent security domains. A second contribution is the definition of cross-layer policy relationships that associate software artifacts, service identities, event permissions, deployment configurations, and runtime evidence. A third contribution is a comparative evaluation model that identifies measurable indicators for future experimental validation without presenting hypothetical improvements as observed results.

The proposed framework is intended as an architectural and methodological contribution. It is not presented as a deployed prototype or as an experimentally validated security product.

2. Literature Review

2.1. Zero-Trust Security Architecture

Zero trust has developed as a response to the limitations of security models that assume users and systems inside a protected network are inherently trustworthy. Rose et al. (2020) describe zero trust as an architectural approach in which access decisions are based on explicit authentication, authorization, and contextual policy rather than implicit trust derived from network location. This approach is applicable to microservices because individual workloads may be created, rescheduled, scaled, or terminated dynamically.

In a distributed service environment, zero trust requires more than user authentication. Service-to-service communication must also be authenticated, and authorization decisions should reflect the specific operation, resource, workload identity, and relevant contextual conditions. NIST's guidance establishes a foundation for this approach, but organizations must translate general zero-trust principles into enforceable policies for event brokers, service meshes, container platforms, and cloud-native identities.

A major architectural challenge is maintaining consistent authorization across providers that expose different identity and policy interfaces. Federation can establish trust relationships between identity domains, but federation alone does not guarantee least privilege. A service that is permitted to assume a federated identity may still receive excessive permissions unless authorization is constrained by workload attributes, resource scope, environment, and intended operation. The proposed framework therefore separates identity verification from resource authorization and requires both to be evaluated throughout service execution.

2.2. DevSecOps and Secure Software Supply Chains

DevSecOps integrates security testing and governance into software delivery pipelines. Rather than relying exclusively on manual reviews or penetration testing before release, it applies automated controls during source-code development, dependency management, compilation, artifact packaging, infrastructure provisioning, and deployment.

Shahin et al. (2017) examined continuous integration and continuous delivery research and identified important engineering challenges associated with automation, testing, and delivery practices. These challenges remain relevant to secure distributed applications because increasing deployment frequency can amplify the consequences of weak or inconsistent security gates.

The Secure Software Development Framework proposed by Souppaya et al. (2022) provides practices for preparing development environments, protecting software, producing well-secured software, and responding to vulnerabilities. The SLSA framework similarly emphasizes supply-chain integrity and provenance as mechanisms for establishing confidence in how software artifacts are produced. These approaches can support verification of source revisions, build processes, dependency inventories, and artifact origins.

However, supply-chain assurance is only one part of event-driven application security. A verified container image does not establish that its workload has appropriate permissions to publish financial transactions or read sensitive event streams. Similarly, a successful source-code scan does not establish that a production event schema rejects malicious payloads. The proposed framework extends lifecycle security by connecting software provenance to deployment authorization and runtime identity policies.

2.3. Security Challenges in Event-Driven Microservices

Event-driven architectures decouple producers and consumers through messages or events. This decoupling improves independent deployment and reduces direct service dependencies, but it also changes the security model. Messages may be retained for extended periods, delivered more than once, replayed after failures, or consumed by multiple independent applications. As a result, message security must address confidentiality, integrity, authorization, ordering assumptions, and application-level processing semantics. Richards (2015) explains the architectural trade-offs associated with microservices, including the operational complexity introduced by distributed components. Kleppmann (2017) discusses distributed data systems and the importance of understanding delivery semantics, consistency, and fault tolerance. These considerations are especially relevant when security controls interact with event retries, consumer offsets, idempotent processing, and cross-service workflows.

Broker authentication and transport encryption are necessary but insufficient. Producers should be authorized to publish only to designated topics or event channels, while consumers should receive access only to the streams required for their business functions. Event schemas should be validated, and sensitive fields should be protected through data minimization, encryption, and controlled retention. Applications handling critical transactions may also require signed event envelopes, unique event identifiers, timestamps, and replay-detection mechanisms.

An additional challenge concerns trust propagation. An event produced by an authenticated service does not automatically remain trustworthy after passing through transformation services, dead-letter queues, retries, or third-party integrations. Consumers must evaluate the event's provenance and integrity while independently validating its content and authorization context.

2.4. Multi-Cloud Security and Policy Consistency

Multi-cloud architectures introduce heterogeneity in identity providers, network configurations, workload runtimes, encryption services, and monitoring systems. This heterogeneity can create gaps when security policies are defined independently for each provider. A deployment that complies with one provider's configuration baseline may violate an organizational requirement elsewhere because resource names, permissions, logging interfaces, and default settings differ.

Cloud security research emphasizes that security responsibilities extend beyond provider infrastructure to include customer-controlled identities, applications, configurations, and data. The Cloud Security Alliance's Cloud Controls Matrix offers a structured set of cloud security control objectives that organizations can use to compare governance and assurance requirements across providers (Cloud Security Alliance, 2021).

Container orchestration and service-mesh technologies can provide consistent enforcement points for workload identity, traffic encryption, and network policy. However, neither technology independently solves all multi-cloud security problems. A service mesh may authenticate a workload connection while leaving event-level authorization or application-level data validation to other controls. Likewise, a centralized policy engine may define organizational rules but depend on provider-specific enforcement adapters to implement them correctly.

Consequently, a practical multi-cloud framework must distinguish between common policy intent and provider-specific implementation. It should maintain a consistent security model without assuming that every cloud exposes identical capabilities or that all controls can be enforced through one universal interface.

2.5. Research Gap

The literature provides established foundations for zero-trust architecture, secure software development, microservices engineering, cloud governance, and distributed event processing. Nevertheless, these areas are frequently implemented as separate architectural disciplines. Zero-trust guidance establishes identity-centric access principles, DevSecOps frameworks emphasize secure delivery, and event-driven design literature addresses messaging and distributed processing. A remaining architectural need is to connect these practices through a unified lifecycle model.

The specific gaps addressed by this study are:

- Lifecycle fragmentation: insufficient integration between source-code assurance, artifact provenance, deployment authorization, and production event access.
- Event-specific trust gaps: incomplete treatment of schema validation, replay detection, topic-level permissions, and event provenance within general DevSecOps pipelines.
- Cross-cloud policy inconsistency: difficulty translating common organizational requirements into enforceable provider-specific controls.
- Limited end-to-end traceability: inadequate linkage between build artifacts, workload identities, event-processing decisions, policy evaluations, and incident evidence.
- Evaluation limitations: the need for a reproducible evaluation model that measures both security effectiveness and the operational costs of continuous enforcement.

The proposed framework addresses these gaps at the architectural level. Its effectiveness must subsequently be assessed through implementation, controlled testing, and comparison with baseline security configurations.

3. Research Methodology

3.1. Research Design

This study adopts a design-oriented conceptual research methodology. The objective is to develop a coherent security architecture from established principles and identify how its components can be evaluated systematically. The methodology is informed by design science research, which emphasizes the construction and assessment of artifacts intended to address relevant organizational problems (Hevner et al., 2004).

The research process consists of problem identification, review of relevant security and architecture literature, extraction of design requirements, framework construction, control mapping, comparative analysis, and development of an evaluation plan. The resulting artifact is a reference architecture rather than a production implementation. No organization-specific dataset, controlled experiment, penetration-test result, or measured performance improvement is claimed.

3.2. Design Requirements

The framework is guided by security, operational, and governance requirements derived from the literature and the characteristics of event-driven multi-cloud systems.

Security requirements include explicit authentication, least-privilege authorization, and encryption in transit, secure secret handling, artifact integrity, event validation, and auditable policy decisions. The architecture must also support credential revocation, vulnerability response, and containment of compromised workloads. Operational requirements include automated pipeline integration, repeatable deployment, scalable event processing, fault tolerance, and compatibility with heterogeneous cloud environments. Security controls should avoid unnecessary synchronous dependencies in high-throughput event-processing paths. Governance requirements include centralized policy definition, controlled exceptions, environment-specific approvals, data classification, traceable evidence, and accountability for security decisions. These requirements are intended to make the framework suitable for organizations operating under internal security policies and applicable regulatory obligations.

3.3. Proposed Framework Components

The architecture contains five interdependent layers.

The first layer, secure development and software supply-chain assurance, incorporates source-code analysis, dependency scanning, secret detection, software bills of materials, artifact signing, and build provenance. Pipeline controls assess whether a candidate release meets defined security criteria before promotion.

The second layer, identity and policy governance, establishes workload identities, federated authentication, least-privilege authorization, policy-as-code, and controlled exceptions. Identity verification and authorization are treated as separate decisions so that successful authentication does not automatically grant unrestricted resource access.

The third layer, event-driven communication protection, secures producers, brokers, consumers, schemas, and event-processing workflows. It incorporates mTLS where appropriate, broker-level access control, schema compatibility validation, event provenance, replay detection, and application-level validation.

The fourth layer, multi-cloud deployment and orchestration, coordinates policy enforcement across container platforms, managed cloud services, network boundaries, and deployment environments. It uses provider-specific adapters while preserving common organizational policy intent.

The fifth layer, continuous observability and adaptive response, correlates security logs, distributed traces, deployment metadata, identity events, broker activity, and runtime alerts. It supports investigation, policy reassessment, credential revocation, workload isolation, and controlled rollback.

3.4. Methodological Workflow

The research methodology follows these steps:

- Identify trust boundaries across development environments, build systems, deployment pipelines, event brokers, workloads, and cloud providers.
- Map each trust boundary to authentication, authorization, integrity, confidentiality, and monitoring requirements.
- Assign controls to the appropriate lifecycle stage and enforcement point.
- Define relationships between software artifacts, workload identities, event permissions, deployment policies, and runtime evidence.
- Compare the proposed framework conceptually with less integrated security approaches.
- Define indicators, baselines, and test scenarios for subsequent empirical evaluation.

The workflow emphasizes that security cannot be assessed exclusively by counting automated tools. A meaningful assessment must determine whether controls are correctly configured, enforced at the appropriate boundary, and capable of preventing or detecting the threats they are intended to address.

3.5. Evaluation Criteria

The conceptual evaluation uses five dimensions: security coverage, policy consistency, event integrity, operational overhead, and traceability. Security coverage concerns the proportion of required controls implemented and tested across lifecycle stages. Policy consistency concerns the degree to which equivalent security requirements are enforced across cloud providers. Event integrity addresses the detection or rejection of unauthorized, malformed, or replayed messages. Operational overhead includes pipeline execution time, authorization latency, resource consumption, and maintenance effort. Traceability concerns the ability to connect a production event or security incident to its originating service, software artifact, deployment, and policy decision. These dimensions provide a basis for quantitative evaluation in a future implementation. In the present study, they are used for structured architectural comparison rather than as measured experimental outcomes.

4. Results and Discussion

4.1. Architectural Overview of the Proposed Framework

The principal outcome of the design process is a five-layer architecture that connects development-time security assurance with identity-centric access control, event-level protection, multi-cloud enforcement, and continuous operational monitoring. The framework is designed to maintain security context as software moves from source code to a deployed service and subsequently participates in asynchronous communication.

The central architectural principle is that trust must be established independently at each relevant boundary. A successful build does not imply that an artifact is safe to deploy, an approved deployment does not imply that a workload may access every event stream, and an authenticated producer does not imply that all messages it generates are valid. Each stage therefore requires its own verification criteria, with evidence passed to subsequent stages where appropriate.

Figure 1 presents the conceptual architecture. The layers are connected through a common governance and observability model, while enforcement remains distributed across the pipeline, identity infrastructure, event brokers, workload runtimes, and cloud-specific security services.

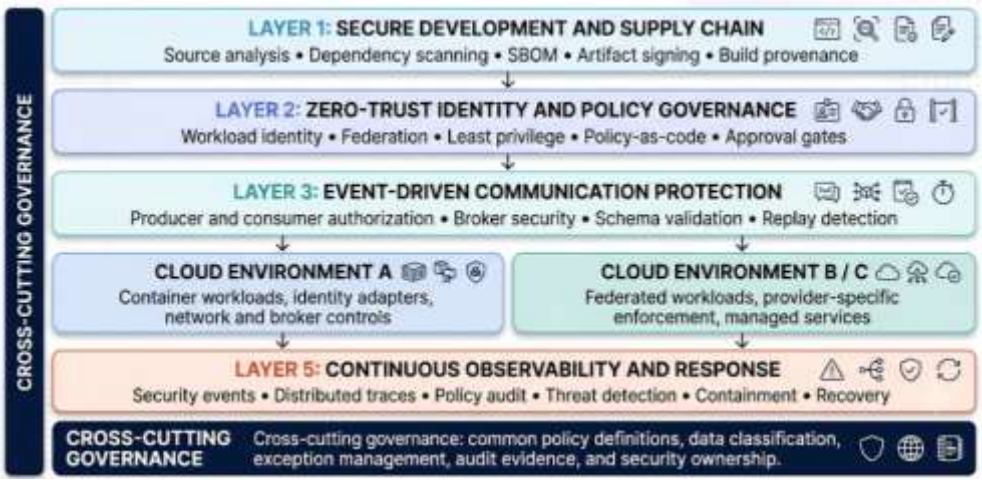


Figure 1. Proposed Zero-Trust DevSecOps Architecture for Event-Driven Microservices across Multi-Cloud Environments

Source: Author's conceptual framework, developed from established zero-trust and secure software development principles.

The diagram represents logical relationships rather than a mandatory deployment topology. In an implementation, observability operates across every layer, not merely at the bottom of the architecture. Similarly, identity and policy governance is a cross-cutting capability even though it is represented as a distinct layer for clarity. This separation allows organizations to assign clear responsibilities while preserving the interdependencies required for end-to-end protection.

4.2. Integration of Zero Trust into DevSecOps Pipelines

The first major finding from the architectural analysis is that zero-trust enforcement must begin before production deployment. Source-code repositories, build agents, artifact registries, infrastructure templates, and deployment controllers are privileged components because they influence the software that eventually receives production access.

A secure pipeline should authenticate developers and automation identities, restrict repository permissions, detect exposed credentials, inspect third-party dependencies, and verify artifact provenance. Build outputs should be traceable to approved source revisions and controlled build processes. Where organizational policy requires it, signed artifacts and attestations should be verified before deployment. These measures align with the secure development practices described by Souppaya et al. (2022).

Security gates should evaluate both the artifact and its intended deployment context. For example, a container image with acceptable vulnerability status might still be rejected if its deployment manifest requests privileged execution, mounts sensitive host resources, or grants unrestricted network access. Similarly, an approved service should not be promoted if its declared event subscriptions exceed the permissions justified by its business function.

A policy-as-code approach can express these conditions in a version-controlled and reviewable form. The pipeline evaluates policy before release, while runtime enforcement checks whether the deployed workload continues to operate within its authorized boundaries. This approach reduces dependence on manual interpretation and helps maintain consistency between development and operations.

Nevertheless, excessive security gating can slow delivery when scans are redundant, policy exceptions are poorly managed, or external services become bottlenecks. Organizations should therefore distinguish between critical violations that block deployment, lower-risk findings that require remediation within a defined period, and exceptional releases that require documented approval. Such classification must not become a mechanism for routinely bypassing mandatory protections.

4.3. Event-Level Identity, Authorization, and Integrity

The second finding is that event-driven systems require security controls beyond conventional service-to-service authentication. A message broker is not merely a transport component; it is a shared authorization boundary through which business events may be distributed to multiple independent services. The framework assigns distinct permissions to producers and consumers. Producers receive permission to publish only to designated topics or channels, while consumers are limited to approved subscriptions and consumer groups. Administrative permissions are separated from ordinary application permissions. Broker-level authorization is combined with application validation so that unauthorized access and malformed payloads can be addressed at different layers.

Event schemas provide an additional security boundary. Validation can reject messages with unexpected structures, invalid types, missing mandatory fields, or incompatible schema versions. However, schema validity alone cannot establish that a business event is authentic or semantically authorized. A structurally valid transaction event might still contain an unauthorized account identifier or a transaction value inconsistent with the producer's permitted role. Consumers therefore need business-rule validation and appropriate data-access checks.

Replay protection is especially important where repeated processing could trigger duplicate payments, repeated provisioning actions, or unintended state changes. Unique event identifiers, bounded timestamp checks, deduplication records, and idempotent consumer operations can reduce this risk. The appropriate mechanism depends on event retention, processing guarantees, latency requirements, and the consequences of duplicate execution. Because distributed messaging commonly permits retries and duplicate delivery under some configurations, replay detection must distinguish malicious repetition from legitimate redelivery.

Cross-cloud messaging also requires careful handling of trust propagation. A service receiving an event should verify the source identity and integrity evidence supported by the messaging architecture, but it should not blindly inherit the producer's permissions. The consumer's own identity and authorization policy should determine which actions are permitted. This separation prevents the event itself from becoming an unrestricted authorization token.

4.4. Multi-Cloud Policy Enforcement and Governance

A third finding is that consistent security governance requires a common policy model combined with environment-specific enforcement. A single global policy definition can express organizational intent, but it cannot eliminate differences in provider identity systems, messaging services, container platforms, or logging interfaces. The proposed framework separates policy authorship from policy enforcement. Central governance defines requirements for workload authentication, permitted event access, encryption, artifact verification, and audit retention. Provider-specific adapters translate those requirements into the controls supported by each environment. Compliance checks then verify the resulting configurations rather than assuming that a successful configuration deployment necessarily means the policy has been enforced correctly.

This distinction is important for avoiding false consistency. Two cloud environments might use different technical mechanisms to achieve equivalent security outcomes. Conversely, identically named settings might provide different guarantees or operate at

different scopes. Effective governance therefore requires testing the behavior of controls and recording the evidence used to establish compliance.

Policy distribution also introduces availability and consistency concerns. A central authorization service may simplify governance but create a dependency that affects deployment or runtime decisions if it becomes unavailable. The framework should distinguish between policies that must be evaluated online and policies that can be securely cached or enforced locally. Cached decisions should have defined expiration periods, revocation behavior, and limits on the actions permitted during connectivity failures. Fail-open behavior must be avoided for sensitive operations unless an explicitly approved risk policy justifies it.

4.5. Continuous Observability and Incident Response

The fourth finding is that zero-trust security becomes operationally useful when authorization and integrity decisions can be connected to observable evidence. Individual alerts may be insufficient to explain how an incident developed across multiple services and cloud environments. A compromised workload, for example, might initially appear as an unusual identity event, followed by unauthorized topic access and then unexpected downstream activity.

The proposed framework correlates identity records, build provenance, deployment metadata, broker audit logs, distributed traces, and runtime security alerts. Correlation identifiers and trace context help investigators reconstruct processing paths, while carefully designed audit records provide evidence of which policy was applied and why access was permitted or denied. Sensitive event payloads should not be copied indiscriminately into logs; observability should favor metadata, classifications, hashes, and controlled references where these are sufficient.

Runtime response can include revoking workload credentials, reducing topic permissions, isolating a compromised service, suspending a deployment, or routing suspicious events for controlled investigation. Automated actions should be proportional to the risk and the confidence of the detection mechanism. A low-confidence anomaly should not necessarily trigger a broad production shutdown, whereas confirmed credential compromise may justify immediate revocation.

Observability also supports feedback into the development lifecycle. If a production incident is traced to an insecure dependency, excessive permissions, or missing schema validation, the finding should inform future pipeline checks and reusable security templates. This creates a continuous improvement cycle rather than treating incident response as an isolated operational activity.

4.6. Comparative Analysis of Security Approaches

Table 1 compares the proposed framework with conventional DevSecOps, perimeter-oriented security, and partially integrated zero-trust approaches. The comparison is qualitative and reflects the intended architectural coverage of each approach, not the measured performance of specific products.

Table 1. Comparative Analysis of Security Approaches for Multi-Cloud Event-Driven Microservices

Evaluation Dimension	Perimeter-Oriented Security	Conventional Devsecops	Partially Integrated Zero Trust	Proposed Framework
Primary security focus	Network boundary and trusted zones	Automated software delivery checks	Identity-centric access control	Lifecycle-wide security integration
Source-code and dependency assurance	Often separate	Commonly integrated	Depends on pipeline integration	Integrated with release authorization
Workload identity	May rely on network location or credentials	Variable	Explicit identity verification	Explicit identity with scoped authorization
Event-broker permissions	May be coarse-grained	Often handled separately	Can be identity-aware	Producer, consumer, topic, and operation controls

Schema and replay protection	Application-dependent	Variable	Not inherently provided	Explicitly included in event security
Cross-cloud governance	Often provider-specific	Pipeline-dependent	Requires federation and policy integration	Common policy intent with provider-specific enforcement
Runtime monitoring	Network and perimeter telemetry	Build and deployment telemetry	Identity and access telemetry	Correlated identity, artifact, event, and runtime evidence
Operational complexity	Moderate initially; gaps may increase at scale	Pipeline maintenance and tool integration	Identity and policy integration overhead	Higher integration and governance requirements
End-to-end traceability	Limited across application boundaries	Stronger for build and release stages	Stronger for access decisions	Designed to link release, identity, event, and incident evidence

Source: Author's qualitative comparison based on the framework's design requirements and the cited literature.

The comparison highlights a distinction between security control availability and security control integration. Conventional DevSecOps can provide excellent source-code and dependency assurance, but those capabilities do not automatically extend to event-level authorization. Likewise, a zero-trust deployment may authenticate every workload connection without establishing the integrity of every event or linking runtime activity to software provenance.

The proposed approach seeks to close these gaps by making lifecycle relationships explicit. Its main advantage is the potential for more consistent enforcement and more complete incident reconstruction. Its main disadvantage is implementation complexity: organizations must maintain policy mappings, identity integrations, event schemas, evidence pipelines, and operational procedures across multiple environments.

This trade-off suggests that the framework is best introduced incrementally. Organizations with limited security engineering capacity may begin with workload identity, least-privilege broker permissions, artifact verification, and essential audit correlation before adopting more advanced event attestation and adaptive response mechanisms.

4.7. Security Evaluation Scenarios and Expected Outcomes

The proposed architecture can be evaluated through controlled scenarios representing realistic attacks and configuration failures. The purpose is to establish whether the framework's controls operate as intended, rather than to assume that the presence of a control guarantees security.

Table 2. Proposed Security Evaluation Scenarios and Assessment Indicators

Test Scenario	Relevant Controls	Assessment Indicator	Expected Security Outcome
Unauthorized producer attempts to publish to a restricted topic	Workload identity, broker authorization	Unauthorized publication acceptance rate	Unauthorized publication is denied
Consumer receives a malformed event	Schema and business-rule validation	Invalid-event rejection rate	Invalid events are rejected or quarantined
Previously processed event is replayed	Event identifiers, deduplication, idempotency	Duplicate side-effect rate	Repeated processing does not produce unauthorized duplicate effects
Container image has a prohibited vulnerability or untrusted provenance	Image scanning, signing, provenance verification	Unauthorized artifact promotion rate	Release is blocked according to policy

Workload requests excessive permissions	Policy-as-code, admission controls, least privilege	Excess-permission detection and prevention rate	Deployment is rejected or permissions are restricted
Federated identity is used outside its approved scope	Identity federation, contextual authorization	Unauthorized cross-cloud access acceptance rate	Cross-cloud access is denied
Compromised service exhibits suspicious behavior	Runtime monitoring, correlation, response policies	Detection time and containment time	Suspicious activity is identified and contained
Central policy service becomes unavailable	Local enforcement, bounded caching, failure policy	Availability and policy-compliance rate during outage	Critical controls remain effective under the approved failure policy

Source: Author's proposed evaluation matrix. These are test scenarios and expected outcomes, not experimentally observed results.

The evaluation should measure both prevention and operational consequences. For instance, a low unauthorized-publication acceptance rate is meaningful only if legitimate publication remains available and the broker can sustain its required throughput. Similarly, faster incident containment should be evaluated alongside false-positive rates, service disruption, and recovery time.

A suitable experiment would deploy representative services across at least two cloud environments and establish a baseline using conventional pipeline scanning and provider-specific access controls. The proposed framework could then be tested against the same workload, threat scenarios, and event volume. Repeated trials would help distinguish reliable security improvements from incidental variation.

Evaluation should also consider the cost of maintaining policy consistency. A framework that prevents attacks but introduces excessive deployment delays or frequent false positives may be difficult to sustain in production. Consequently, security effectiveness, service reliability, developer productivity, and operational cost should be analyzed together.

4.8. Practical Implications and Limitations

The framework has practical implications for organizations that operate distributed applications under strict security and governance requirements. Security teams can use the layered architecture to define control ownership, platform engineering teams can implement reusable deployment and identity templates, and development teams can incorporate event validation and least-privilege permissions into service contracts.

The architecture may also support compliance evidence collection by associating policy definitions with pipeline results, artifact attestations, deployment records, and runtime audit events. Such evidence can improve auditability, but automated records do not independently establish regulatory compliance. Organizations must still interpret applicable obligations, validate control effectiveness, and maintain appropriate oversight.

Several limitations must be acknowledged. First, the study presents a conceptual framework and has not implemented or benchmarked it against a live multi-cloud system. Second, the effectiveness of individual controls depends on their configuration, implementation quality, and integration with the application. Third, the architecture does not remove all supply-chain risks, compromised identities, insider threats, or vulnerabilities in trusted components. Fourth, centralized governance can create operational dependencies if policy distribution and enforcement are not designed for failure conditions.

Finally, the framework does not assume that all event payloads require cryptographic signatures or that every authorization decision must contact a central service. These choices should be based on data sensitivity, threat models, performance constraints, and operational risk. The architecture provides a decision structure within which those trade-offs can be documented and evaluated.

5. Conclusion

This research proposed a Zero-Trust DevSecOps Framework for Event-Driven Microservices across Multi-Cloud Environments to address the limitations of fragmented security practices in distributed cloud-native applications. The framework integrates software

supply-chain assurance, workload identity, policy-as-code, event-level protection, multi-cloud enforcement, and continuous runtime monitoring into a unified lifecycle architecture. Its central principle is that trust must be established and continuously reassessed at every significant boundary rather than inferred from network location, successful deployment, or previously authenticated communication.

The architectural analysis demonstrates why securing event-driven microservices requires a broader approach than conventional source-code scanning or network protection. Event brokers introduce shared access boundaries, asynchronous communication complicates incident reconstruction, and cross-cloud federation can create unintended permission paths. The proposed framework addresses these concerns through narrowly scoped producer and consumer permissions, schema and business-rule validation, replay resistance, verified software artifacts, and correlated operational evidence. Integrating these mechanisms can provide a more consistent basis for preventing unauthorized actions and investigating security incidents.

The comparative discussion also identifies an important trade-off. Greater integration can improve policy consistency and traceability, but it introduces additional engineering, governance, and operational requirements. Effective implementation therefore depends on risk-based prioritization, reusable security controls, well-defined exception procedures, and careful management of enforcement dependencies. Security controls should be proportionate to the sensitivity of the data and the consequences of unauthorized processing.

The principal contribution of this study is a conceptual architecture and evaluation framework that connects zero-trust principles with the distinctive requirements of event-driven DevSecOps. Because the study does not report implementation or experimental measurements, its proposed benefits should be understood as architectural expectations rather than demonstrated performance improvements. The framework establishes a foundation for future empirical work examining whether integrated enforcement can improve security outcomes while preserving the availability, scalability, and delivery speed required by modern multi-cloud applications.

6. Future Scope

Future research should implement the framework in a representative multi-cloud environment using container orchestration, workload identity federation, an event-streaming platform, and automated CI/CD pipelines. Such an implementation would enable controlled comparison with conventional DevSecOps and less integrated zero-trust baselines. Researchers could measure unauthorized access acceptance, vulnerability-gate effectiveness, event-processing latency, deployment duration, policy inconsistency, false-positive rates, and incident containment time under repeatable conditions.

A second research direction concerns adaptive authorization. Machine-learning techniques could help identify unusual event access patterns, unexpected producer-consumer relationships, and deviations from established service behavior. However, adaptive decisions should remain constrained by explicit security policies, with human approval for high-impact exceptions and deterministic enforcement for critical operations. The reliability of these techniques would need to be evaluated against adversarial behavior, changing workloads, and false-positive risks.

Third, future studies could investigate privacy-preserving security observability across organizational and cloud boundaries. Selective metadata collection, redaction, retention controls, and privacy-aware correlation could help investigators reconstruct incidents without unnecessarily exposing sensitive business data. Research is also needed into interoperable event-provenance mechanisms that remain verifiable as messages pass through transformation services, queues, and third-party integrations.

Finally, the framework could be extended to incorporate confidential computing, workload attestation, software supply-chain risk scoring, and automated policy conformance testing. Longitudinal evaluations would be particularly valuable because they could reveal whether the architecture remains maintainable as cloud services, dependencies, threat patterns, and regulatory requirements evolve. Such research would help establish the practical conditions under which zero-trust DevSecOps improves security without imposing disproportionate operational costs.

The following conceptual workflow illustrates how event security and software delivery assurance interact from development through runtime response.

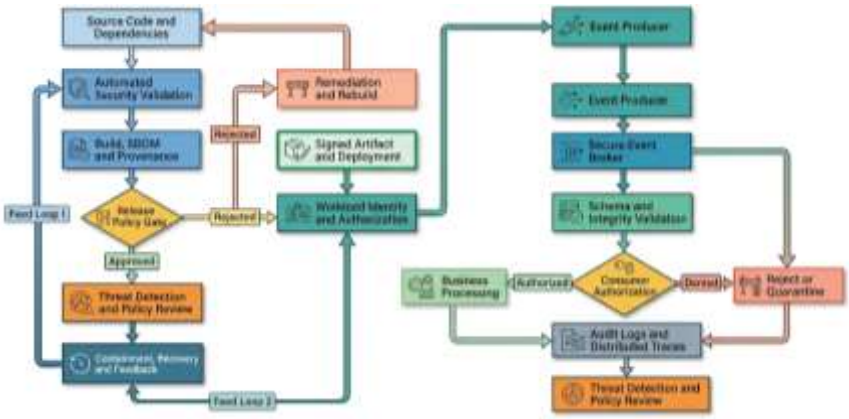


Figure 2. Secure Event Lifecycle and Continuous DevSecOps Feedback

Source: Author's conceptual representation of the proposed framework. SBOM refers to a software bill of materials.

The workflow emphasizes that security validation is not completed at deployment. The operational evidence generated during event processing informs policy review, incident response, and subsequent development cycles. Rejected releases return to remediation, while suspicious runtime behavior can trigger access restrictions, containment, and revised security checks.

References

[1] The following references use APA 7th edition style and provide foundational sources for zero trust, secure software development, microservices, continuous delivery, and cloud security governance.
[2] Cloud Security Alliance. (2021). Cloud Controls Matrix (CCM) v4.0. https://cloudsecurityalliance.org/research/cloud-controls-matrix
[3] Arora, A. S., Yachamaneni, T., & Kotadiya, U. (2023). Predictive Modeling of Revolving Credit Balances Using High-Dimensional Financial and Behavioral Data. International Journal of AI, BigData, Computational and Management Studies, 4(1), 98-107.
[4] Virendra Jangid. (2023). *AI-Driven Optimization of Regulatory Reporting Systems in Global Banking*. International Journal of Science and Applied Technology, 14(4). https://www.ijsat.org/papers/2023/4/11259.pdf
[5] Marella, B. C. C. (2025). AI and XR in Supply Chain: Revolutionizing Sustainable Practices for a Better Tomorrow. In *Exploring the Impact of Extended Reality (XR) Technologies on Promoting Environmental Sustainability* (pp. 83-98). Cham: Springer Nature Switzerland.
[6] Hosahalli Seenappa, V. K. (2025). Next-generation intelligent multi-cloud enterprise networking: Integrating secure architecture, AI-optimized traffic engineering, and LLM-driven autonomous network operations. *International Journal of Leading Research Publication (IJLRP), 6*(12), 1–11.
[7] Vegineni, G. C. (2025). Sustainable Tech Development: Integrating AI, XR, and CI/CD Pipelines for a Greener Future. In *Exploring the Impact of Extended Reality (XR) Technologies on Promoting Environmental Sustainability* (pp. 267-293). Cham: Springer Nature Switzerland.
[8] Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS Quarterly, 28(1), 75–105. https://doi.org/10.2307/25148625
[9] Marella, B. C. C. (2022). Driving Business Success: Harnessing Data Normalization and Aggregation for Strategic Decision-Making. *International Journal of Intelligent Systems and Applications in Engineering, 10*(2), 308-317.
[10] Gajula, S., & Margam, M. (2026, February). A secure and scalable cloud-based banking service model leveraging AI and advanced cyber security. In 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC) (pp. 1-5). IEEE.
[11] Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero trust architecture (NIST Special Publication 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207
[12] Yachamaneni, T., Arora, A. S., & Kotadiya, U. (2025, August). Reinforcement Learning (RL) for Dynamic Credit Limits Using Java and Cloud-Based AI Services. In International Conference on Computing and Communication Networks (pp. 434-453). Cham: Springer Nature Switzerland.
[13] Kumar, A., Sridhar, R., Dhenia, R. N. K., & Kanani, I. J. K. (2025, July). Advancing Security: Novel Attack Detection Techniques for Industrial Internet of Things Datasets. In *2025 3rd World Conference on Communication & Computing (WCONF)* (pp. 1-5). IEEE.
[14] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629
[15] Virendra Jangid. (2024). Cloud-Native Financial Data Platforms for Risk and Compliance Management. International Journal of Intelligent Systems and Applications in Engineering, 12(23s), 4389 –. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/8349
[16] Gajula, S. (2025). Federated intelligence in financial ecosystems: A privacy-preserving AI framework for cross-border risk analysis. Journal of Information Systems Engineering and Management, 10(60s).

[17] Vasantha Kumar Hosahalli Seenappa. (2025). Predictive Configuration Governance for AI-Assisted Network Automation in Hyperscale MPLS/IP Backbone Networks. *International Journal of Intelligent Systems and Applications in Engineering*, *13*(2s), 352–358. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/8534

[18] Souppaya, M., Scarfone, K., & Dodson, D. (2022). Secure software development framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities (NIST Special Publication 800-218). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-218

[19] Verma, A., Madan, J., Sridhar, R., Grover, V., Singh, K., & Shrivastav, N. (2025, September). Influence of Temperature and Layer Thickness on the Photovoltaic Efficiency of (FA) 2 BiCuI 6-Based Perovskite Solar Cells. In *2025 2nd International Conference on Circuits, Power and Intelligent Systems (CCPIS)* (pp. 1-5). IEEE.

[20] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70–93. https://doi.org/10.1145/2898442.2898444

[21] Marella, B. C. C. (2023). Scalable Generative AI Solutions for Boosting Organizational Productivity and Fraud Management. *International Journal of Intelligent Systems and Applications in Engineering*, *11*(10), 1013-1023.

[22] Gadam, H., Yadava, A. K., Arora, A. S., Yachamaneni, T., & Kotadiya, U. (2025, July). Revolutionizing Wealth Management with Temporal Attention Mechanism Based Convolutional Network for Algorithmic Stock Trading Strategies. In 2025 3rd International Conference on Data Science and Network Security (ICDSNS) (pp. 1-4). IEEE.

[23] Virendra Jangid. (2025). *AI-Powered Cross-Border Securities Processing and ADR Platform Optimization*. International Journal of Leading Research Publication, 6(11). https://doi.org/10.70528/IJLRP.v6.i11.2233

[24] Verma, A., Madan, J., Singh, K., Gothwal, S., Shrivastav, N., & Sridhar, R. (2025, September). Toward Clean Energy: Carrier Density Optimization in FAMASnGeI 3-Based Perovskite Solar Cells. In *2025 2nd International Conference on Circuits, Power and Intelligent Systems (CCPIS)* (pp. 1-5). IEEE.

[25] OWASP Foundation. (2023). OWASP API security top 10—2023. https://owasp.org/API-Security/

[26] Vegineni, G. C. (2025, October). Interactive Design Playbook: Visual Guides and Component Libraries for Adaptive Web Portals. In *International Conference on Artificial Intelligence and Networking* (pp. 534-544). Cham: Springer Nature Switzerland.

[27] Hosahalli Seenappa, V. K. (2026). *Machine learning based computer device for cloud computing security* (UK Registered Design No. 6538224). UK Intellectual Property Office.