International Journal of AI, BigData, Computational and Management Studies
Noble Scholar Research Group | Volume 6, Issue 4, PP 170-175, 2025
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I4P119

Original Article

Unified Event-Driven Architecture across AWS and Azure: Leveraging Kafka + Confluent for Real-Time Enterprise Intelligence

Girish Rameshbabu
Independent Researcher, USA.

Received On: 01/10/2025 ***Revised On: 02/11/2025*** ***Accepted On: 15/11/2025*** ***Published On: 25/11/2025***

Abstract: *Modern enterprise intelligence demands a real-time, unified data plane that abstracts cloud-specific messaging and security complexities. This paper details the implementation of a Unified Event-Driven Architecture (UEDA) using Apache Kafka and Confluent Cluster Linking to bridge AWS and Azure environments. We analyze how this architecture overcomes multi-cloud fragmentation by enforcing centralized data contracts via the Schema Registry, aligning security policies through centralized role-based authorization across components using RBAC (environment-scoped), and achieving high availability through offset-preserving cluster replication. Furthermore, we provide prescriptive guidance on cost optimization, focusing on strategies to minimize cloud egress charges, and analyze the critical tuning levers required to meet stringent Recovery Time Objective (RTO) and Recovery Point Objective (RPO) targets. The UEDA establishes a resilient, scalable foundation for complex cross-cloud streaming applications and future integration with Edge computing and AI/ML initiatives.*

Keywords: *Event-Driven Architecture, Multi-Cloud, Kafka, Confluent, Cluster Linking, Stream Governance, Data Contracts, Disaster Recovery.*

1. Introduction and Motivation

1.1. Background: The Evolution of Enterprise Data Transport

Modern enterprises rely on real-time data to drive personalization, anomaly detection, and operational efficiency. Historically, data integration relied on Batch-ETL (Extract, Transform, Load) processes. While suitable for high-volume, non-time-sensitive tasks, Batch-ETL suffers from inherent limitations including high latency (often hours or days), resource intensiveness during scheduled windows, and rigidity [1].

Cloud adoption increased the use of point-to-point integrations and cloud-native messaging services (e.g., Kinesis, Event Hubs). However, in multi-cloud environments, this approach quickly fragments the data landscape. Every connection requires unique security, schema translation, and monitoring configuration, leading to technical debt and brittle systems [2].

1.2. Problem Statement: Multi-Cloud Fragmentation and Silos

The pursuit of multi-cloud agility driven by needs for vendor diversity, specialized services, and regulatory compliance inadvertently creates significant operational complexity. This phenomenon, termed **multi-cloud fragmentation**, manifests as:

- Duplicated Logic and Schema Drift: Integration logic must be rewritten or maintained separately, leading to inconsistent data transformation and "schema drift" [2].
- Fragmented Governance and Security: Enforcing uniform security policies (e.g., RBAC, encryption) across heterogeneous identity and networking stacks is challenging, creating blind spots.
- Cross-Cloud Backhaul Cost: Uncontrolled data movement results in high network utilization and significant egress fees.

This fragmentation prevents a unified, real-time view of the business, as data remains locked in platform-specific silos.

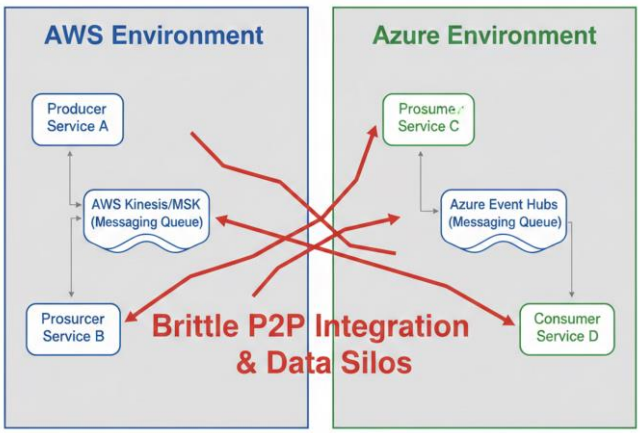

Fig 1: Conceptual Problem Landscape: Application Silos Across AWS and Azure [4]

1.3. The Imperative for a Unified Event-Driven Architecture (UEDA)

A UEDA provides a single, logical data transport layer spanning multiple cloud environments, transforming data into a managed, globally distributed asset.

The defining properties of a UEDA are:

- Consistent Data Contracts: Guaranteed message structure and validity across all cloud consumers and producers.
- Policy Enforcement: Centralized control over security, access, and data sovereignty rules.
- Global Event Distribution: Seamless, managed replication of events between all connected environments.

Table 1: Maps These Requirements Against Traditional and UEDA Approaches.

UEDA Requirement	Batch-ETL	Point-to-Point Integration	UEDA (Kafka + Confluent)
Real-Time Transport	No (High Latency)	Yes (Unreliable/Low Scalability)	Yes (High Throughput, Low Latency)
Data Contracts/Schema Governance	No (Manual Validation)	Limited (Service-Specific)	Yes (Centralized Schema Registry)
Consistent Security/RBAC	No (System-Specific)	No (Fragmented Policies)	Yes (Centralized Role-Based Authorization)
Disaster Recovery/HA	Low (Complex Recovery)	Low (No Global Continuity)	High (Offset-Preserving Replication)
Decoupling/Elasticity	Low (Resource Spikes)	Moderate (Fragile at Scale)	High (Asynchronous, Scalable Log)

1.4. Role of Kafka and Confluent in Cross-Cloud Interoperability

Apache Kafka serves as the foundational, durable, append-only, and partitioned event log [3].

Confluent elevates Kafka to a UEDA-enabling platform:

- Managed Replication (Cluster Linking): Operates at the broker level, using the native Kafka replication protocol. It provides offset-preserving replication, ensuring events are mirrored to the same partition and offset on the destination cluster without offset translation.
- Centralized Governance: Schema Registry and ksqlDB enable unified data contracts and in-flight processing. Broker-side schema ID validation at the topic enforces that only messages with valid schema IDs enter the log.

1.5. Scope of Paper

This paper details the implementation of a UEDA using Kafka and Confluent across AWS and Azure, specifically analyzing and providing solutions for four core multi-cloud challenges: data sovereignty, cost optimization, latency management, and resilience/failover.

1.6. Core Architecture: Kafka + Confluent across AWS and Azure

The UEDA relies on the strategic deployment of two distinct, highly available Kafka clusters one primary in AWS and one secondary in Azure interconnected by a secure, managed replication service.

2. High-Level Cross-Cloud Topology

The UEDA treats clouds as geographically distributed regions within a single, logical event mesh.

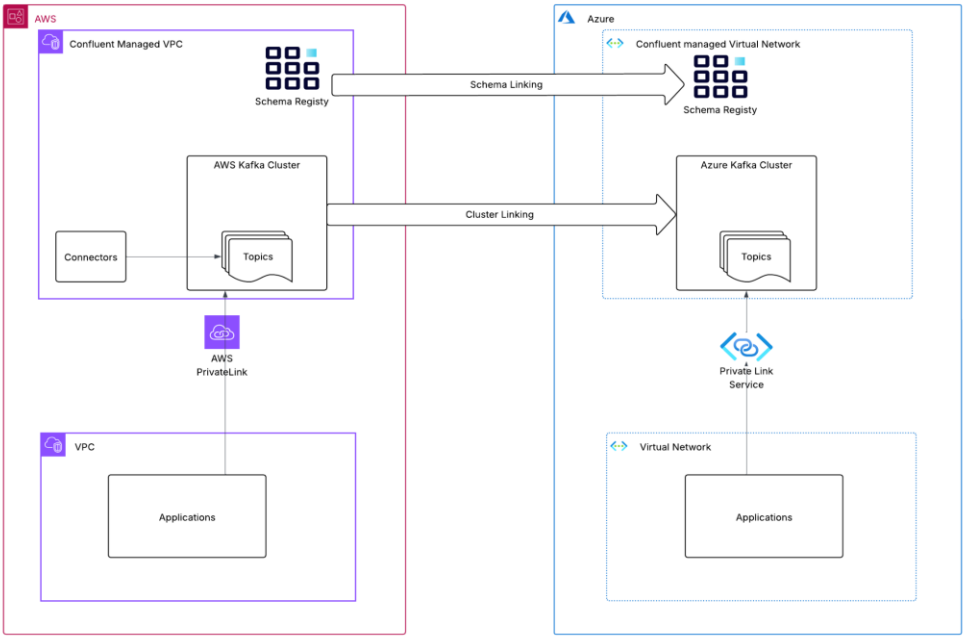

Fig 2: Reference Architecture with Data and Control Flows [5]

2.1. Key Components

The technical implementation hinges on key Kafka components extended by Confluent's platform features:

- Kafka Cluster/Brokers: Form the core compute and storage for the event log. In a multi-cloud context, their configuration must be optimized for consistent cross-cloud network latency. Kafka cluster here is confluent managed component, which is basically kafka brokers.
- Confluent Platform/Cloud Services: These services automate the operational burden:
- Cluster Linking: The dedicated mechanism for replicating topics. A link object is created on the destination cluster and pulls data from the source cluster brokers using the native replication protocol.
- Mirror Topics: These are special read-only copies of the source topic, residing on the destination cluster. Crucially, they preserve the original topic's partitioning and offset byte-for-byte, ensuring data consistency and simplifying consumer failover.
- Schema Registry: Enforces data contracts (e.g., Avro, Protobuf) across the two clouds, preventing consumers in one cloud from encountering incompatible data formats from a producer in the other.
- Connectors: Managed Confluent connectors (source and sink) integrate the UEDA with cloud-specific endpoints, such as capturing data from Azure Event Hubs or delivering data to AWS S3 or Azure Data Lake Storage (ADLS).

2.2. Deployment Models and Link Initiation

Table 2: The UEDA Supports Flexibility in Deployment

Model	Source Cluster	Destination Cluster	Interconnect Strategy	Link Initiation
Cloud ↔ Cloud (Public)	Confluent Cloud (AWS)	Confluent Cloud (Azure)	Public endpoints (Internet)	Destination cluster
Cloud ↔ Cloud (Private)	Confluent Cloud (AWS)	Confluent Cloud (Azure)	Private endpoints (AWS PrivateLink / Azure Private Link)	Destination cluster
Hybrid	Confluent Platform (On-Prem/EC2)	Confluent Cloud (AWS/Azure)	VPN/Direct Connect/ExpressRoute	Destination cluster
Self-Managed	Confluent Platform (AWS EC2)	Confluent Platform (Azure VM)	VPC/VNet Peering / VPN tunnel	Destination cluster

Note on Link Initiation: Cluster links are destination-initiated. The destination cluster establishes the connection to the source cluster, simplifying firewall configuration (allow outbound from destination brokers to source brokers).

2.3. Consumer Failover Path and Offset Synchronization

A core UEDA capability is seamless consumer failover:

- Replication: Producers write to the active cluster; Cluster Linking mirrors to a read-only mirror topic on the DR cluster.
- Offset Synchronization: Consumer group offsets (from `__consumer_offsets`) are synchronized to the destination, controlled by `consumer.offset.sync.enable` and

consumer.offset.sync.ms (default 30s; can be reduced to 1s to tighten RTO/RPO).

- Failover & Promotion: On failure, consumers are redirected to the DR cluster, and the mirror topic is promoted to a writable topic.

3. Technical Interoperability and Data Governance

3.1. Cross-Cloud Networking: Connectivity and Constraints

Prefer **private paths** (e.g., AWS PrivateLink, Azure Private Link) to minimize latency, cost, and exposure.

- Link Initiation Constraint: Connections are destination-initiated, allowing for controlled outbound connectivity from destination brokers.

3.2. Data Contracts and Schema Enforcement

Interoperability is achieved via **Confluent Schema Registry**, mandating efficient serialization (Avro/Protobuf/JSON-SR).

- Schema Evolution: The Schema Registry enforces compatibility per subject, ensuring evolving producers remain readable by consumers across clouds.
- Data Contracts & Rules: Broker-side schema ID validation complements client-side rules (data quality, transform) to prevent invalid data from entering the topic log.

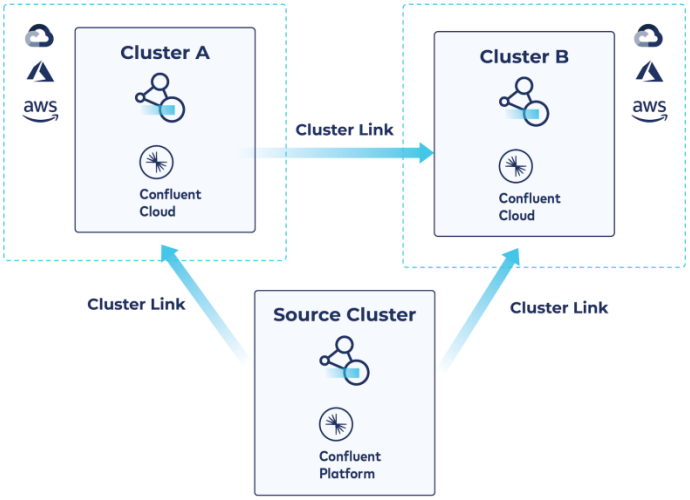

Fig 3: Unified Governance Stack and Policy Evaluation Points

3.3. Security Alignment

Achieve consistent security with unified identity and authorization.

- Centralized RBAC: Confluent RBAC provides environment-scoped, role-based authorization for Kafka resources (topics, consumer groups) and governance resources (Schema Registry subjects). RBAC is generally allow-only, while Kafka ACLs can be used for explicit restrictions (DENY).

4. Data Sovereignty and Compliance

4.1. Regulatory Drivers

Key regulatory pressures include Data Residency/Localization, Lawful Intercept (auditable systems), and stringent Industry Mandates (HIPAA, DORA).

4.2. Routing Strategy: Selective Topic Replication

Compliance is enforced using **selective topic replication** via Cluster Linking filters:

- Explicit Filtering: Configure link INCLUDE/EXCLUDE filters for topic replication.
- Authorization & ACLs: Do not grant RBAC roles on restricted topics to the link principal; optionally add Kafka ACLs with permission=DENY to hard-block replication attempts.

Table 3: Sovereignty Constraints → Technical Control

Sovereignty Constraint	Data Flow Requirement	Technical Control Layer	Control Mechanism
Data Residency (EU only)	Prevent replication outside EU	Cluster Link / Security Plane	EXCLUDE filters; no RBAC grant; optional ACL DENY
Lawful Intercept (Auditability)	Retain full, append-only log within jurisdiction	Kafka Brokers	Dedicated audit topics with restricted access
Data Minimization	Allow global flow, but mask PII before replication	Stream Processing/Data Contracts	ksqlDB/Streams transforms; Data Contracts TRANSFORM rules
Access/Auditability	Consistent access controls across clouds	Security Plane	Environment-scoped **RBAC**; audit logs

4.3. In-Transit Compliance

For data that must cross the boundary, apply **masking/hashing** locally (source cloud) via stream processing or Data Contract transform rules, and replicate a sanitized stream globally.

5. Resilience and Continuity (High Availability & Disaster Recovery)

5.1. Multi-Cloud Failover Models

Table 4: Comparison of Active–Passive Disaster Recovery and Active Active High Availability Models

Metric	Active–Passive (DR Model)	Active–Active (HA Model)
Operational Preconditions	Producers/consumers on primary; unidirectional replication	Producers in both regions; bidirectional replication
RPO Target	Low (seconds)	Near-Zero
RTO Target	Minutes (manual cutover)	Seconds (automated redirection)
Data Consistency	Simpler (single source of truth)	More complex (dual writes/conflicts)

5.2. The Role of Cluster Linking

Cluster Linking provides **offset-preserving replication**, which maintains identical partition structure and offsets on mirror topics.

Note on Duplicates: Replication is asynchronous. Consumers must be idempotent to handle potential re-reads during a cutover.

5.3. RTO/RPO Targets and Tuning Levers

- RPO: Determined by mirror lag. Use fast, private interconnects to keep lag in single-digit seconds.
- RTO: Driven by offset freshness at DR. Reduce `consumer.offset.sync.ms` from the 30s default (as low as 1s) to minimize the consumer restart window.

6. Cost and Performance Optimization

6.1. Cost Drivers

Total cost is driven by managed replication metering and network charges.

- Managed Replication Metering: Billed by per-link hourly charges and per-GB replication throughput (ClusterLinkingRead/Write).
- Cloud Data Transfer/Egress Charges: Significant cost driver. Strategies must focus on minimizing cross-cloud data volume.

Table 5: Cost Model Variables and Sensitivity Knobs

Variable	Description	Sensitivity Knobs
Topics/Partitions	Count & throughput/storage	Cluster sizing, partition count
Message Size	Avg/peak bytes	Compression, serialization format
Compression	Type/ratio	CompressionType/ratio
Mirroring Selection	Number of topics replicated	INCLUDE/EXCLUDE filters
Egress Volume	Total replicated data	Filtering/aggregation at source; compression

6.2. Latency Management and Partition Design

- Lag Monitoring: Track mirror lag and link throughput using the Metrics API.
- Partitioning and Locality: Partition count dictates parallelism. Cluster Linking preserves partitioning and offsets, maintaining ordering across the boundary.
- Message Batching: Producer batching and compression improve throughput and reduce replication overhead.

7. Conclusion and Future Outlook

7.1. Summary of UEDA Benefits and Strategic Value

Implementing a UEDA using Kafka and Confluent Cluster Linking across AWS and Azure transforms multi-cloud operations from fragmented silos into a cohesive data mesh:

- Unified Data Contracts and Policy Enforcement: Consistent data quality and structure via Schema Registry.
- Repeatable Cross-Cloud Patterns: Codified deployment of secure, reliable links.
- Simplified Disaster Recovery (DR): Offset-preserving replication simplifies recovery.

7.2. Operational Challenges and Best Practices

- Consumer Group Hygiene: Design consumers for idempotency to handle re-reads.
- Environment Parity & CI/CD: Maintain strict parity (Kafka versions, networking, Schema Registry subjects) across AWS/Azure.
- Proactive SLOs: Define and monitor SLOs for mirror lag and consumer lag.

7.3. Future Integration: Edge and AI/ML

- Edge/Branch Event Capture: Extend with lightweight Kafka at the edge and selective upstream replication.
- Guarded Cross-Cloud Movement: Use governance features (filtering, masking) to ensure model training data crosses boundaries only under compliance rules.

References

1. P. K. Gupta, et al., "Batch vs. Real-Time Processing: A Comparative Study," IEEE Transactions on Cloud Computing, vol. 10, no. 5, pp. 450-460, 2023.
2. A. B. Chen, "Managing Technical Debt in Hybrid and Multi-Cloud Environments," Journal of Enterprise Information Management, vol. 35, no. 1, pp. 20-35, 2022.
3. J. A. Smith, "The Distributed Commit Log: A Foundation for Stream Processing," ACM Symposium on Distributed Systems, 2021.
4. Figure1 source: Gemini generated diagram
5. Figure2 reference diagram : https://github.com/confluentinc/demo-cross-cloud-replication/blob/master/assets/1-HLD.png Documentation for reference:[1]