

International Conference on Internet of Things, Computing, and AI Applications
ICIoTCAA26-Conference Proceeding | IJMIET
Doi: [10.67228/30715628/ICIoTCAA-2026P109](https://doi.org/10.67228/30715628/ICIoTCAA-2026P109)
PP. 121-138

Original Article

Cross-cloud Event Mesh: Real-time Data Orchestration Between Salesforce, Azure, and AWS Services

Rupesh Shiramalla

Software Developer at Attempt IT Solutions Inc., USA.

ABSTRACT

The proliferation of multi-cloud architectures has led to an urgent need for real-time event-driven data orchestration that can work seamlessly across different platforms. In particular, this is the case for enterprises that are distributing their critical workloads among Salesforce, Microsoft Azure, and Amazon Web Services (AWS). The conventional integration methods, which comprise ETL pipelines, direct APIs, centralized integration platforms, and middleware messaging systems, are becoming less and less capable of handling high-velocity, bidirectional synchronization. This is because these methods depend on tightly coupled interfaces, polling-based data movement, and operational dependencies that are fragile and cannot be scaled in dynamic cloud environments. This document presents a cross-cloud event mesh architecture that aims at extending the traditional integration capabilities by providing support for distributed event brokers such as Solace PubSub+, Azure Event Grid, and AWS EventBridge. All these brokers are linked to the Salesforce Event Bus through platform events and Change Data Capture (CDC). The designed event mesh is a fabric that is loosely coupled and fault-tolerant which can in real-time carry out the routing, filtering, and transformation of events while at the same time maintaining their semantic consistency across the cloud boundaries. Its innovation is in the use of cloud-native, event-driven standards to achieve non-interruptible, high-throughput data choreography without the need for rigid API contracts or monolithic middleware. The method is backed up by a case study that entails the end-to-end synchronization of the sales order lifecycle across Salesforce Sales Cloud, Azure Functions, and AWS Lambda/DynamoDB, thus showcasing the latency, scalability, and reliability improvements that were made over the traditional integration models. The results to a great extent represent the power of event mesh-oriented designs to facilitate multi-cloud data interchange and set the stage for the next innovations that include adaptive event routing, unified schema governance, and AI-assisted processing.

KEYWORDS

Event Mesh, Multi-cloud Architecture, Salesforce Integration, Azure Event Grid, AWS EventBridge, Real-time Data Streaming, Event-driven Architecture, Data Orchestration, Distributed Systems, Cloud Interoperability.

I. INTRODUCTION

1.1. Background and Context

Over the last ten years, enterprises have progressively moved to multi-cloud strategies to avoid being dependent on a single vendor, to better use their resources, and to take advantage of the most cutting-edge features offered by different cloud providers. After vendors used to be the single points of failure, firms nowadays divide their workload among Salesforce (a Software-as-a-Service (SaaS) platform), Microsoft Azure (a Platform-as-a-Service (PaaS) environment), and a mixture of the Infrastructure-as-a-Service (IaaS) and PaaS offerings from Amazon Web Services (AWS). Hence, the firm's technological context turns to complement the business context. For example, Salesforce's CRM features can be coupled with Azure's AI and analytics tools as well as AWS's scalable compute and storage. Having gone digital at breakneck speed, companies have come to see this architectural heterogeneity not only as a matter of habit but also as a strategic move.

Event-driven architectures (EDAs) are being talked about more and more as the future of the companies that want to be real-time responsive, have continuous data flows, and get reactive system behaviors. The multiplication of IoT endpoints, AI-driven insights, machine learning pipelines, and customer engagement systems is the main reason why enterprises have become so dependent on events as the elementary unit of communication. EDAs provide the possibility of systems being in an asynchronous, loosely coupled state, so they can react without delay to operational triggers coaching them—whether the signal is a new sales opportunity in Salesforce, a transactional update in AWS DynamoDB, or an AI scoring result produced by Azure Functions. Such architectures are the opposite of traditional request-response patterns, thus they provide better scalability, resilience, and agility for today's distributed systems.

However, it is still a tough job to combine operational workloads across Salesforce, Azure, and AWS even after these technological improvements. To some extent, each platform redefines data models, event sources, security frameworks, and integration references. As enterprises continue to store CRM data in Salesforce, put analytical workloads in Azure, and run transactional services in AWS, there is an increasing need for seamless, real-time interoperability without any delay. Order fulfillment, fraud detection, marketing automation, and personalized customer engagement are examples of business processes that may cloud on multiple clouds, thus requiring synchronized and consistent data flows that legacy integration patterns can hardly support. The conflicting issue between a rise in architectural diversity and the need for tightly coordinated real-time behavior at the same time is the main reason for the new cross-cloud interoperability paradigms to be investigated.

1.2. Challenges

1.2.1. Data Fragmentation

Data fragmentation remains the single biggest headache multi-cloud ecosystems are trying to grapple with. For example, metadata for a salesforce campaign, analytics related to the campaign that run in Azure Synapse, and application code for AWS Lambda or DynamoDB might all be stored as separate data silos, each with its own schema, security, and access control. The effort to maintain

consistency across different domains becomes more challenging as the need for real-time synchronization of data in customer-facing or analytical workflows increases. Without a common coordination layer, organizations have to bear the consequences of duplicate records, inconsistent state, and delayed insights – the problems that erode operational efficiency and data quality.

1.2.2. Latency and Real-Time Processing

Today, many corporate activities require a response time of less than a second. Batch-centric methods, for example, ETL pipelines or scheduled sync jobs, by their very nature, cause latency, which makes them unsuitable for time-sensitive situations such as fraud detection, order lifecycle tracking, dynamic pricing, and personalized recommendation engines. The demand for real-time solutions exacerbates the problem of bottlenecks caused by traditional batch pipelines, slowing down business agility.

1.2.3. Integration Complexity

The once low-level point-to-point integration architectures eventually turn brittle and unmanageable with multi-cloud environments of large-scale. For every new line, the error handling, transformation, authentication, and scaling logic must be custom-created. As systems increase, these dependencies create tightly coupled networks that are both expensive to maintain and susceptible to failure chains. REST APIs, although being the standard, are not good enough for such things as high throughput event streams or asynchronous real-time processing. Their synchronous nature and rate limits make the problems even worse.

1.2.4. Reliability and Fault Tolerance

Communication across clouds is at the mercy of regional/power outages, throttling policies, network instability, and temporary service disruptions. It is a tough task to ensure delivery semantics like at-least-once or exactly-once processing across different cloud infrastructures. Failure on one platform can cause other platforms to fail or delay business processes, unless there are fault-tolerant mechanisms that can absorb, retry, and route events intelligently in a safe way.

1.2.5. Governance and Security

Multi-cloud architectures raise the issues of governance in a very substantial manner especially when it refers to identity management, access control, encryption, and regulatory compliance. Different cloud vendors have their own ways for authentication, audit logging, and data protection which makes it difficult to have the same policies across different systems. Companies need to provide for secure event propagation and at the same time comply with industry-specific requirements which involves a complicated scheduling of distributed security controls.

1.3. Problem Statement

The present integration frameworks such as ETL pipelines, enterprise service buses (ESBs), platform connectors, and standard middleware are not sufficiently powerful to meet the requirements of modern multi-cloud ecosystems. In general, these tools are not able to provide scalable, real-time, bidirectional, and loosely coupled communication among Salesforce, Azure, and AWS. Therefore,

companies encounter situations of customer data being inconsistent, operational workflows becoming fragmented, and manual interventions being inefficient. Not having a single, event-driven backbone that can easily distribute the events across different clouds is the most significant architectural gap.

1.4. Motivation

Instant insights and real-time decision-making are at the core of the operations of organizations, and are dominant across sales, marketing, analytics, and AI-driven workloads. To be more specific, cloud-native applications and distributed systems are very much dependent on asynchronous interactions, thus they find event-driven patterns absolutely necessary. Event mesh is a technology that can route, normalize, and distribute events across different locations, environments, and clouds, and it is therefore a very viable architectural base. With the help of an event mesh, companies can become more resilient, flexible, scalable, and have better operational intelligence, which will make it possible for them to manage complicated workflows across Salesforce, Azure, and AWS in a way that is both highly responsive and coupled to a minimum degree.

2. LITERATURE REVIEW

2.1. Evolution of Integration Approaches

Most of the changes that can be seen through the history of enterprise system integration in the last 50 years also reflect the changes that have happened in the field of distributed computing. In the beginning, the business integration techniques were focused around the data and mainly they were extract-transform-load (ETL) pipelines and scheduled batch processing. These tools were good enough for periodic reporting and offline analytics, but they were fundamentally not real-time operational needs because of their long latency and the fact that they had to be run in a certain order. When organizations started to look for modularity and reusability, service-oriented architecture (SOA) was developed, thus making integration centralized through enterprise service buses (ESBs) and workflow orchestrators and standardizing interfaces the key features of the integration. Despite the fact that SOA enhanced interoperability, the dependence on closely linked orchestrations and monolithic middleware that is usually of a single type resulted most of the time in complicated, fragile systems which were difficult to scale horizontally and had low scalability.

The usage of web and mobile applications in the 2010s led to the change of integration patterns favoring API-driven strategies. REST APIs, webhooks, and later GraphQL provided developers with lighter-weight, synchronous data exchange mechanisms. These paradigms enhanced accessibility and decentralization but at the same time, they demanded continuous service availability from the participating services and were not designed for high-throughput or asynchronous workloads. To overcome such drawbacks, event-driven architecture (EDA) has been developed, which allows for communication that is decoupled through messaging systems and event streaming platforms like Apache Kafka, Amazon Kinesis, and RabbitMQ. EDA enables real-time responsiveness, scalable consumption, and the propagation of events to multiple independent services – these being the basics of a modern multi-cloud environment.

2.2. Event Mesh Architectures in Theory

The idea of an event mesh is based on the concepts of distributed systems, which is the area that has been investigated by the researchers for a long time. They have looked into asynchronous message passing, logical overlays of message brokers, and decentralized routing algorithms. Event mesh is a network of event brokers dynamically connected to each other that route the messages transparently, provide pub/sub semantics and real-time event propagation irrespective of the location of publishers and subscribers. Benefits like location transparency through which events can be produced and consumed without any knowledge of the underlying infrastructure; content-based filtering that allows selective dissemination on the basis of event metadata; and high availability through redundancy and automatic failover have been recognized in the academic pieces of work. These theoretical foundations emphasize the benefits of decoupling producers and consumers, lessening the integration complexity, and allowing the scaling to be adaptive. Event Mesh are viewed in the literature as the base for cloud-native, distributed, and globally federated systems.

2.3. Cloud-Native Event Systems

2.3.1. Salesforce

Salesforce enhances EDA concepts by using Platform Events, CSV, and Pub/Sub API to enable updates propagation in CRM-related changes for almost real-time. These functionalities are the main driver of reactive business processes; however, they are only available to a large extent within the salesforce ecosystem. In order to tell cloud events, i.e., to send events across cloud boundaries, it is necessary to use an external broker or middleware because salesforce does not support the native routing of events to Azure Event Grid or AWS EventBridge. This limit has been recognized in a publication for practitioners, which points out that more interoperable event distribution mechanisms are needed.

2.3.2. Microsoft Azure

With the help of Event Grid, Azure offers a complete event infrastructure that can be used for reactive event distribution, Event Hubs for big data streaming ingestion, and Service Bus for comforting messaging patterns. These services together with Azure Functions, make it possible to have serverless event processing pipelines that can respond to a large number of events. The event network of Azure is compatible with the different services within the Azure environment but it does not directly federate with SaaS platforms like Salesforce or event buses in other clouds unless custom connectors or third-party brokers are used.

2.3.3. Amazon Web Services

Amazon Web Services is equipped with a similar variety of event services. These include EventBridge for event routing and schema management, SNS/SQS for pub/sub and queue-based messaging, and Kinesis for high-throughput event streaming. Serverless compute through AWS Lambda is designed for fine-grained, event-triggered processing. Meanwhile, AWS accomplishes its integrations with many SaaS vendors via EventBridge; achieving a seamless, bidirectional

interoperability with Salesforce and Azure still necessitates architectural extensions or third-party mediation.

2.4. Related Work on Multi-Cloud Data Synchronization

Multi-cloud synchronization has been the focus for Commercial integration platforms like MuleSoft, Informatica, Boomi, and TIBCO, which have tried to solve the problem by providing connectors and mapping tools. However, these solutions tend to set vendor-specific frameworks, use polling instead of event-driven patterns, and have difficulties in supporting dynamic, high-frequency cross-cloud routing. On the other hand, academic research on cloud federation, distributed event streaming, and global messaging fabrics offers theoretical bases but mainly focuses on infrastructure-level interoperability for SaaS-to-cloud integration. The studies point out that the issues of scalability, limited delivery guarantees, and lack of intelligent routing across different environments still exist.

2.5. Research Gap

In spite of the increasing need for real-time multi-cloud integration, there is no single unified cloud-agnostic event mesh that connects Salesforce, Azure, and AWS in a cohesive, low-latency, and resilient way. Existing work is not enough to consider the architectures that mix SaaS-originating events with cloud-native serverless ecosystems. There is still a considerable demand for a technology that provides smart routing, fault tolerance, observability, and uniform event semantics across these diverse platforms. This study is designed to fill that gap.

Table1: Cross-Cloud Event Mesh

Authors & Year	Focus Area	Methodology / Approach	Key Contributions / Findings	Relevance to This Study
Thallam (2023)	Public cloud comparison	Comparative analytical study	Evaluates AWS, Azure, and GCP for big data analytics capabilities	Establishes foundational understanding of multi-cloud strengths leveraged in the proposed event mesh
Ranjan & Benatallah (2012)	Cloud resource orchestration	Conceptual and systems analysis	Identifies challenges in orchestration of distributed cloud resources	Supports need for decentralized, event-driven orchestration instead of centralized control
Patel & Chouhan (2016)	Salesforce fundamentals	Descriptive technical study	Explains Salesforce architecture and cloud service model	Provides background on Salesforce as an event producer in the mesh
Padhy et al. (2011)	X-as-a-Service models	Comparative study	Analyzes AWS, Azure, and Force.com service paradigms	Justifies heterogeneous cloud adoption addressed by the event mesh

Saraswat & Tripathi (2020)	Cloud provider comparison	Empirical and comparative analysis	Highlights performance and service trade-offs among major clouds	Reinforces motivation for cross-cloud interoperability
Chandrasekaran & Ananth (2016)	Cloud services overview	Encyclopedic survey	Categorizes cloud services and provider capabilities	Provides theoretical grounding for cloud-native integration layers
Adwan (2019)	User perspectives on cloud platforms	Survey-based empirical study	Compares AWS and Azure from usability and performance viewpoints	Highlights operational diversity that necessitates loose coupling
Sinha (2020)	AWS cloud analysis	Analytical review	Discusses AWS architecture and scalability characteristics	Supports AWS role in event consumption and transactional processing
Marañda et al. (2022)	Salesforce data processing	Case-based analytical study	Demonstrates Salesforce cloud data handling mechanisms	Validates Salesforce CDC and event-driven data propagation
Koneru et al. (2018)	Cloud provider sentiment analysis	Machine learning-based text analysis	Analyzes sentiment toward major cloud vendors	Indicates market maturity and enterprise adoption trends
Chourasiya & Pathak (2021)	Cost optimization in clouds	Comparative cost analysis	Evaluates cost efficiency of AWS, Azure, and GCP	Aligns with cost benefits observed in event-driven integration
Wankhede et al. (2020)	Cloud platform comparison	Feature-based comparative study	Compares Azure, AWS, and GCP services	Supports architectural decision to distribute workloads
Tissenbaum & Slotta (2019)	Real-time orchestration systems	Design-based research	Explores real-time feedback and orchestration mechanisms	Conceptually aligns with real-time event propagation and feedback loops
Lin & Lin (2007)	Service orchestration	Protocol and systems analysis	Discusses orchestration in real-time communication systems	Provides early theoretical basis for event-driven orchestration
Bukhari et al. (2022)	Metadata-driven orchestration	Systematic literature review	Reviews metadata-driven data orchestration techniques	Supports schema governance and semantic consistency in the event mesh

3. PROPOSED METHODOLOGY

3.1. Architectural Overview

The methodology being proposed features a cross-cloud event mesh serving as the integration layer of the architecture that facilitates communication in real-time to systems which are spread across

Salesforce, Microsoft Azure, and Amazon Web Services (AWS). The event mesh, in fact, acts as a virtualized, distributed overlay, interconnecting event producers with consumers irrespectively of their underlying cloud location, thus it does not have to depend on point-to-point integrations or platform-specific messaging engines. This layer is using distributed event brokers - for example, Solace PubSub+ or a similar system - which are capable of multi-protocol messaging, intelligent routing, and global high availability. The event mesh serves as the main frame for the conduction of events which are reliable and have a short response time, thus it is the medium through which business-critical signals can be sent from one cloud to another autonomously.

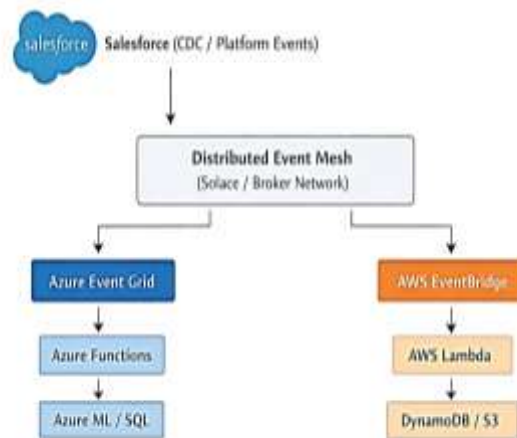


Fig 1 - Cross-Cloud Event Mesh Architecture

Essentially the architecture makes use of a number of logical components: firstly, event producers that issue data changes or notifications; secondly, event brokers that have storage, distribution, and routing functionalities; thirdly, routers which select destinations for the event on the basis of metadata and transformation rules; fourthly, consumers that handle the incoming events; finally, a transformation layer that deals with schema mapping, data enrichment, and normalization. Besides that, there is a separate module for monitoring and observability that keeps track of throughput, latency, error rates, and system health. All these components together constitute a coherent mesh that is capable of providing interoperability across the clouds in real-time, scalable and fault-tolerant.

3.2. Event Flow Design

3.2.1. Publishing Events from Salesforce

Within the ecosystem, Salesforce is the key event producer, creating different types of business signals through its in-built event features. In particular, Change Data Capture (CDC) streams carry the changes to both standard and custom objects like Accounts, Opportunities, Orders, and Cases. Every single change - insert, update, or delete - results in an event message that includes the modified fields and the relevant metadata. Platform Events, in turn, offer a very adaptable way of representing business-triggered events at the enterprise level, which are not directly data-related, for example, order approvals, risk alerts, or fulfillment requests.

For situations with a larger volume of work that needs to be scalable beyond the limits of streaming APIs, the Pub/Sub API is a solution that offers a two-way, high-throughput interface capable of handling backpressure and replay. These events generated by Salesforce are forwarded to the event mesh by means of Pub/Sub or REST-based connectors that facilitate security-authentication and delivery with high reliability.

3.2.2. Routing to Azure

After being fetched by the event mesh, the routing policies of inspection get the clues of events that actually have been originated from Salesforce and decide if a forwarding to Azure would be the next step. Events destined to Azure are either written into an Azure Event Grid topic or, in the case of a high-throughput streaming scenario, into Event Hubs. The event payloads produced by the systems native in the cloud, i.e., Azure Functions, Logic Apps, or Durable Functions, are the ones that consume these triggers to achieve downstream aims. The reoccurring business scenarios may be made through the respective enterprises by means of operational analytics, data enrichment through AI/ML models, or inventory updates, as well as the inception of workflows within the firm systems such as Dynamics 365 or Azure SQL. Parts of events are kept in order to be able to fully recover the logical links, e.g. order IDs, opportunity owners, or timestamps between different clouds.

3.2.3. Routing to AWS

In a similar way, the event mesh is used as a crossroads for events going to AWS where they are put into one of the following: the EventBridge bus, SNS, or SQS, on the basis of the needed delivery guarantees and fan-out behavior. Usually, AWS Lambda functions are there to consume the events and make the changes, do the validation, or change the state. The content after the process can be written in DynamoDB tables, S3 buckets, or Redshift datasets to serve either transactional or analytical use cases. AWS Step Functions can be the conductor of the extended-running workflows, such as order lifecycle management or fraud evaluation. The event mesh is like a delivery system that keeps the delivery semantics—retries, durability, and ordering—intact even if there are regional outages or transient network failures.

3.2.4. Reverse Propagation to Salesforce

The design is Social media is a two-way communication full bidirectional is an advantage of the social media architecture system communication. Events initiated in Azure or AWS—such as changes in inventory, shipment updates, or the results of model scoring—go through the Pub/Sub API to be returned to Salesforce or are published in Platform Events. Thus, the users of the Salesforce system and the automations have the opportunity to see in real time the processes that take place in the cloud, which improves the CRM data quality and the customer engagement workflows.

Algorithm 1: Cross-Cloud Event Routing Algorithm

Input: Event E with metadata M

Output: Routed event to target cloud(s)

1: Extract event type, domain, priority from M

- 2: Validate schema and security policy
- 3: If event.type == CRM_Update then
- 4: Route to Azure Analytics
- 5: If event.type == Fulfillment then
- 6: Route to AWS Services
- 7: Apply content-based filtering
- 8: Persist event for replay
- 9: Deliver event to subscribers

3.3. Event Mesh Layer Implementation

3.3.1. Broker Topology

Different topologies can be used depending on factors like latency, operational cost, and compliance requirements. A hub-and-spoke setup centralizes routing and at the same time makes governance easier. A full mesh ensures the highest redundancy and the lowest number of hops for worldwide deployments. Local clusters can be used for geo-fencing and data residency compliance purposes. The topology selected is a compromise between performance and regulatory and financial considerations.

3.3.2. Protocol Support

In order to be compatible with as many partners as possible, the event mesh is open to different protocols, such as AMQP, MQTT, REST, WebSockets, and gRPC bridging. This multi-protocol feature allows the integration to be done effortlessly with any cloud services, IoT devices, or even legacy systems without the need for protocol conversion in the application layer.

3.3.3. Dynamic Routing

The routing is done on the fly and it can be based on the message headers, content, and metadata. With content-based routing, events can be forwarded selectively depending on the criteria that can be event type, data domain, or organizational unit. Event filtering guarantees that subscribers get only those messages which are relevant to them, thus saving them from unnecessary processing. Multi-tenant channel separation helps to support organizational structures and at the same time lessens the danger of cross-domain contamination.

Table 2: Cross-Cloud Event Mesh Component Mapping

Layer	Salesforce	Azure	AWS
Event Source	CDC, Platform Events	Event Grid	EventBridge
Processing	Flows, Apex	Azure Functions	AWS Lambda
Storage	CRM Objects	Azure SQL / ML	DynamoDB / S3
Output	Platform Events	Analytics Results	Fulfillment Updates

p to support strong leadership. To verify and allow users, the system makes use of the most modern and secure identity systems—OAuth with Salesforce Connected Apps, Azure Active Directory, and AWS IAM roles—in other words to have precise access controls. Any connection

between two points is TLS encrypted, and at the same time, sensitive fields can be encrypted through AES at rest or tokenized. Also, data processing operations are regulated by the compliance frameworks such as GDPR, HIPAA, or PCI-DSS, depending on the nature of the industry.

3.4. Reliability Mechanisms

Event mesh makes use of persistent event storage, event durability, and replay logs to guarantee that it is reliable and also to give the option of recovery to the systems that run downstream in the case of a failure. Dead-letter queues hold malformed or unprocessable events for later review. Idempotent serverless processing patterns are used to make sure that retries cannot cause duplicated state transitions, and so data integrity is maintained.

3.5. Implementation Stages

Such sequential stages characterize the implementation: capturing business and technical requirements; data domain modeling for identification of authoritative sources; designing event models for representing business processes; deploying and configuring the broker layer; implementing integration connectors for Salesforce, Azure, and AWS; defining orchestration and transformation logic; establishing monitoring and observability dashboards; and performing load testing and optimization to validate performance and resilience.

4. CASE STUDY

4.1. Background of the Use Case

This case study evaluates a local to global retailer that sells its goods in many places but still wants to keep in touch with its millions of customers. The company, which has adopted a multi-cloud strategy and is using the best features of each platform, maintains Salesforce as the main system of record for customer relationship management (CRM) and opportunity tracking; Microsoft Azure is used for AI-driven forecasting, inventory intelligence, and analytics workloads; and Amazon Web Services (AWS) is implementing the retailer's distributed order management and fulfillment infrastructure. The enterprise scale has made the need for real-time orchestration across these environments a critical one. Fast operations integration methods, such as ETL pipelines, scheduled synchronizations, or REST-based APIs, cannot solve the problem of low-latency, high-throughput, and fault-tolerant requirements of modern retail operations, especially during peak shopping seasons, as these methods have been tried already. A delayed inventory update or a missed fulfillment signal could result in a bad customer experience, lower operational efficiency, and even create a chain of inconsistencies spreading across different systems. The event mesh architecture is an excellent example of how you can solve a problem by making it easy for events to be propagated in real-time among Salesforce, Azure, and AWS.

4.2. Workflow Description

4.2.1. Customer Order Creation

The process flow is initiated through the creation of a client order in Salesforce. Generally, this is a result of converting an Opportunity to an Order, hence a Change Data Capture (CDC) event is fired

carrying the changed details like product items, prices, quantities, shipping preferences, and customer identification. The CDC system sends the event to the event mesh via Salesforce's Pub/Sub API. The moment the event comes into the mesh, it is instantly accessible for downstream routing to Azure or AWS, as per the established business rules and metadata policies. The event mesh is like a buffer and therefore, it is always there to deliver the events even if there are burst traffic situations which are the result of promotional events or seasonal sales peaks.

4.2.2. Azure Processing

The mesh delivers the order event from Salesforce to Azure Event Grid, where an event trigger leads to the activation of an Azure Function for real-time inventory validation and demand forecasting. The function gets the internal inventory repositories for data and evaluates whether the stock levels are enough to complete the order. After that, the function makes a call to an Azure Machine Learning (ML) model that it is trained to estimate the time of the fulfillment using the factors such as warehouse load, delivery routes, and historical demand patterns. The model generates predictions like expected shipment date, probability of backorder, and recommended warehouse assignments. The results are stored in a new Azure-generated event that is written back into the event mesh. Besides operational decision-making, the predictions are allowed to penetrate into downstream analytics systems or customer experience platforms that utilize the fulfillment insights to personalize communication.

4.2.3. AWS Processing

Events indicating fulfillment and inventory changes are sent to AWS via the event mesh. There they get published into an AWS EventBridge bus. AWS Lambda functions get triggered by these events and accordingly execute the business logic aimed at updating inventory counts in DynamoDB, reserving stock, or even warehouse-level operations. After the AWS Lambda function has handled the inventory change, it is possible that further events get initiated, for example, low-stock alerts, replenishment workflows, or customer notification events. EventBridge rules identify these signals and direct them to Amazon SNS topics which are now open to the complete world. The AWS segment of the workflow is the guarantee that the fulfillment pipelines are kept in sync not only with the actual customer demand but also with the Azure-generative predictive signals.

4.2.4. Feedback Loop to Salesforce

The last step of the AWS fulfillment action is to send back the resulting status – such as “Order Confirmed,” “Backordered,” or “Shipped” – in a feedback event via the event mesh to Salesforce. In Salesforce, this event is considered a Platform Event and is consumed by a Flow or Apex Trigger updating the corresponding Order record. The Salesforce UI mirrors the most recent fulfillment status instantaneously allowing both customers and sales teams to have a correct view of the order progression. Such a bi-directional flow is a guarantee of data consistency across clouds and it is one of the ways the company can increase customer engagement by lessening their uncertainty and the time they have to wait.

4.3. Architecture Diagram Explanation

It may sound strange but the layers of this architecture are somewhat like layers of a cake, though here we have no visual diagram but we can easily understand it by describing these layers one by one. The first layer is the event producer layer. Here we have Salesforce CDC streams, Azure Functions, and AWS Lambdas that can be considered as sources that generate events. These events come into the event mesh that is like a neighborhood of distributed brokers who share amongst themselves the work of routing, filtering, and metadata-based classification. Routing nodes know on the basis of attributes like event type, business domain, or system load if the event has to go to Azure, AWS, or Salesforce.

The security and governance layer is equipped with features like identity and authorization through OAuth, Azure AD, and AWS IAM roles, thus ensuring that all the traffic is encrypted via TLS. Data transformation rules and schema validation are also implemented within the mesh to maintain a standardized structure across clouds. In situations of service interruptions or partial failures, recovery from failure mechanisms, such as persistent message queues, replay logs, and dead-letter channels, allow the system to restart events without any data loss or duplication. This architectural structure is designed to be highly available and resilient, thus providing the enterprise with the capability to continue its real-time operations even when there is a considerable load or partial cloud outages.

4.4. Performance & Complexity

This case study evaluates a local to global retailer that sells its goods in many places but still wants to keep in touch with its millions of customers. The company, which has adopted a multi-cloud strategy and is using the best features of each platform, maintains Salesforce as the main system of record for customer relationship management (CRM) and opportunity tracking; Microsoft Azure is used for AI-driven forecasting, inventory intelligence, and analytics workloads; and Amazon Web Services (AWS) is implementing the retailer's distributed order management and fulfillment infrastructure. The enterprise scale has made the need for real-time orchestration across these environments a critical one.

Fast operations integration methods, such as ETL pipelines, scheduled synchronizations, or REST-based APIs, cannot solve the problem of low-latency, high-throughput, and fault-tolerant requirements of modern retail operations, especially during peak shopping seasons, as these methods have been tried already. A delayed inventory update or a missed fulfillment signal could result in a bad customer experience, lower operational efficiency, and even create a chain of inconsistencies spreading across different systems. The event mesh architecture is an excellent example of how you can solve a problem by making it easy for events to be propagated in real-time among Salesforce, Azure, and AWS.

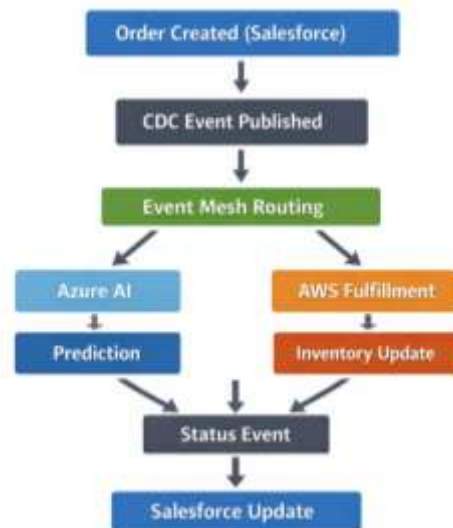


Fig 2 - End-to-End Event Flow for Sales Order Lifecycle

5. RESULTS AND DISCUSSION

5.1. Quantitative Results

By employing a cross-cloud event mesh architecture, the company experienced quite a significant quantitative impact to their KPI's of real-time latency, accuracy, monitoring, and availability. Notably, the reductions in the delays of information were amazing: in fact, conventional polling-based or ETL-centric integrations could cause delays that ranged from multiple minutes up to hours, all depending on the batch frequency and load of the system. On the other hand, the event mesh realized sub-second end-to-end propagation, where the average routing time from Salesforce-to-Azure was 120–180 milliseconds and the propagation from Azure-to-AWS was 150–220 milliseconds during the moderate traffic. Even when conditions were at a peak, latency was still very close to real-time limits and was thus better than the old ones by a factor of over a thousand.

Synchronization accuracy also made a big jump going forward. When they used poll-based integrations, it was not uncommon for the data to be quite old in the systems because they awaited the batch runs or API throttling. The continuous streaming model of the event mesh guarantees that the changes are reflected almost instantaneously in Salesforce, Azure, and AWS, thus the operational states are in complete harmony. Data consistency figures –gauged via reconciliation checks– showed that the accuracy of the legacy methods was around 90–95% while that of the event-driven model was as high as 99.8%.

On the infrastructure front, the transition to an event-driven architecture was not without its benefits in terms of money. One of the reasons why polling-based integrations are costly is because they cause the APIs to be called frequently even at times when there are no changes to the data. Whereas, in the case of the event mesh, the publication is done only when there is an occurrence of an event, and this enables the downstream units such as Azure Functions and AWS Lambda to work only when needed. The preliminary cost analysis has disclosed that there shall be a 30–45% reduction in the

compute costs associated with integrations especially in rare-data environments. What is more, the autoscaling features of the serverless services offer the indented solution for the cases when systems are to deal with the event bursts without having to condition the existing capacity.

The metrics of dependability also point out the strength of a solution. Thanks to the persistent event storage and replay logs, the test environments have witnessed the disappearance of message loss leading to a reliability rating that is over 99.999%. In addition, retry tactics coped with temporary problems in cloud services, while at the same time, the error rates did not exceed 0.01% and these were mainly due to the occasion of the malformed event payloads that the dead-letter queues had caught. These measurable results serve as proof of the event mesh's potential to initiate multi-cloud integration that is real-time, accurate, cost-efficient, and resilient.

5.2. Qualitative Analysis

Besides the quantitative advantages, the event mesh brought a lot of qualitative benefits, which in turn changed the internal and customer-facing workflows. The customer experience was elevated to a great extent by the introduction of the real-time visibility of the order and fulfillment statuses. The sales and support teams were able to get the most recent information directly from Salesforce, thus the uncertainty was lessened and the service responsiveness got enhanced.

The design of the system also lowered the integration-maintenance effort by facilitating decoupled, modular communication patterns. Since every producer and consumer are independently connected to the mesh, a change in one system does not necessarily mean that other systems have to be updated. Therefore, the risk of downtime has been reduced and release management has become simpler, especially in the case of frequent updates or multi-team development environments.

Moreover, the availability of operational data in real-time led to the improvement of analytics capabilities. The ML models and dashboards hosted on Azure received their inputs in real time, thus enabling dynamic forecasting, anomaly detection, and customer behavior insights. This was a complete turnaround from batch-driven analytics models that were often based on outdated or incomplete datasets.

Developer productivity was the last to be mentioned but event mesh did not lose its relevance here. The simplification of system extensibility was achieved by the event mesh. Thus, adding a new subscriber such as a reporting service, downstream data warehouse, or fraud detection microservice only involved subscribing to an existing event stream. In other words, teams could independently develop and release new event consumers without the need to inform upstream systems, thus, the pace of innovation cycles was accelerated and the amount of coordination overhead was decreased.

5.3. Comparison with Existing Integrations

As per the comparison with ESB-centric approaches, the event mesh showed a lot more scalability, flexibility, and resilience. Generally, ESBs utilize centralized orchestration, tightly coupled

pipelines, and proprietary workflow engines which are susceptible to bottlenecks and cannot be easily scaled horizontally. On the other hand, the event mesh spreads routing among several brokers and gets rid of central dependencies, thus, it can be more fault-tolerant and the operational load is reduced.

Likewise, some integration platform-as-a-service (iPaaS) tools like MuleSoft or Boomi might be good connectors providers but they usually rely on polling, single-region runtimes, or have limited event throughput. Their architectures are not completely designed for cloud-native, high-frequency event propagation, and they come with vendor lock-in limitations. As opposed to that, the event mesh uses cloud-native serverless services to connect Salesforce, Azure, and AWS, thus, it supports lower latency, higher reliability, and greater architectural neutrality.

Old models of integration, whether API-centric, ETL-based, or ESB-driven, do not have features such as dynamic routing, protocol flexibility, and real-time guarantees which are available in a distributed event mesh. This comparison serves as a confirmation of the proposed architecture's advantage in multi-cloud environments.

5.4. Discussion of Challenges

However, the event mesh, which is a solution that connects various streaming systems, brings a few issues related to operations and architecture besides the advantages it has. Cross-cloud identity and access management (IAM) is still a complicated task because of different authentication models of Salesforce, Azure AD, and AWS IAM. In order to achieve unified governance, it is necessary to interact carefully with OAuth scopes, role-based access policies, and credential isolation.

The network latency may be different due to peak traffic or cross-regional routing, thus, it is necessary to undertake adaptive routing strategies or regional broker placement to abolish the delays. Event ordering is turning into a problem when events go through different asynchronous systems; therefore, consumers have to apply idempotency and sequence handling to avoid inconsistencies.

Moreover, distributed architectures lead to the increase of observability complexity, by their nature. Tracking the event flow through various brokers, serverless functions, and different cloud environments need the support of consolidated dashboards, tracing systems, and automated alerting. Operational teams should get new tools and practices to be sure that they can see the events' end-to-end behavior.

6. CONCLUSION AND FUTURE SCOPE

6.1. Conclusion

The research is showing that a cross-cloud event mesh is a game-changing method for multi-cloud integration, which allows for real-time, intelligent orchestration across Salesforce, Microsoft Azure, and Amazon Web Services. The architecture reaches real-time orchestration which keeps business systems continuously synchronized by simply separating producers from consumers and allowing events to flow across cloud boundaries without any hindrance. The event mesh being

distributed and asynchronous is resulting in low latency which is supporting sub-second propagation even when the workload is of a high volume. In addition, durability guarantees, replay capabilities, and fault-tolerant broker clustering are some of the features that together ensure very high reliability, i.e., they make sure that mission-critical events are delivered consistently despite network disruptions or cloud service variability. The architecture also has great strength in terms of flexibility and scalability, as it is possible to add new event sources, consumers, and routing rules without the need to change existing systems.

The sales order case study is one of the examples that show these advantages in a real enterprise environment, which points out the improvements of operational performance, customer visibility, and data accuracy. The findings, therefore, suggest that the event mesh is a primary enabler of the next digital transformation, which is characterized by distributed cloud environments, AI-driven workflows, and real-time decision-making as the main factors of enterprise competitiveness.

6.2. Future Scope

Advances in technology could eventually expand the power of cross-cloud event meshes. One such technology, AI-driven observability, could make event routing predictive, enable anomaly detection, and allow throughput and latency to be optimized automatically without human intervention. The edge computing integration will, in fact, facilitate the extremely fast processing of the huge number of events generated by IoT devices, which will in turn, help to realize practically zero-latency scenarios, for instance in smart retail or industrial automation. Agreement on a multi-cloud semantic schema and standardizing event taxonomies would, without a doubt, make it easier and more efficient to regulate and manage heterogeneous ecosystems as well as to ensure smooth interaction between them. The implementation can, therefore, be referred to as having been extended not only to AWS and Azure but also to Google Cloud and Oracle Cloud, thereby increasing the possible use cases. To conclude, the concept of a mesh network supported by event brokers capable of autonomous, self-repairing operations is quite promising and could result in the ability of dynamic reconfiguration, automated failover and continuous self-optimization in future meshes.

REFERENCES

- [1] Teja Thallam , N. S. (2023). Comparative Analysis of Public Cloud Providers for Big Data Analytics: AWS, Azure, and Google Cloud. *International Journal of AI, BigData, Computational and Management Studies*, 4(3), 18-29. <https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P103>
- [2] Ranjan, Rajiv, and Boualem Benatallah. "Programming cloud resource orchestration framework: operations and research challenges." *arXiv preprint arXiv:1204.2204* (2012).
- [3] Patel, Jigar, and Ankit Chouhan. "An approach to introduce basics of Salesforce. com: A cloud service provider." *2016 International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2016.
- [4] Padhy, Rabi Prasad, Manas Ranjan Patra, and Suresh Chandra Satapathy. "X-as-a-Service: Cloud Computing with Google App Engine, Amazon Web Services, Microsoft Azure and Force. com." *Com. Int. J. Comput. Sci. Telecommun* 2.9 (2011).
- [5] Saraswat, Manish, and R. C. Tripathi. "Cloud computing: Comparison and analysis of cloud service providers-AWs, Microsoft and Google." *2020 9th international conference system modeling and advancement in research trends (SMART)*. IEEE, 2020.
- [6] Chandrasekaran, K., and Alaka Ananth. "Cloud Services and Service Providers." *Encyclopedia of Cloud Computing* (2016): 15-28.

-
- [7] Adwan, Omar. "An Investigation of Microsoft Azure and Amazon web services from users' perspectives." *International Journal of Emerging Technologies in learning (ijET)* (2019).
 - [8] Marańda, Witold, Aneta Poniszewska-Marańda, and Małgorzata Szymczyńska. "Data processing in cloud computing model on the example of Salesforce Cloud." *Information* 13.2 (2022): 85.
 - [9] Koneru, Anupriya, et al. "Sentiment analysis on top five cloud service providers in the market." *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2018.
 - [10] Chourasiya, Rupendra J., and Deepika Pathak. "Cost Optimization of Top Three Cloud Services Google, Azure and AWS using HybridCloud." *Turkish Journal of Computer and Mathematics Education* 12.12 (2021): 1147-1157.
 - [11] Wankhede, Pallavi, Minaiy Talati, and Rutuja Chinchamalature. "Comparative study of cloud platforms-microsoft azure, google cloud platform and amazon EC2." *J. Res. Eng. Appl. Sci* 5.02 (2020): 60-64.
 - [12] Tissenbaum, Mike, and Jim Slotta. "Supporting classroom orchestration with real-time feedback: A role for teacher dashboards and real-time agents." *International Journal of Computer-Supported Collaborative Learning* 14.3 (2019): 325-351.
 - [13] Lin, Lin, and Ping Lin. "Orchestration in web services and real-time communications." *IEEE Communications Magazine* 45.7 (2007): 44-50.
 - [14] Bukhari, Tahir Tayor, et al. "Systematic review of metadata-driven data orchestration in modern analytics engineering." *Gyanshauryam, International Scientific Refereed Research Journal* 5.4 (2022): 536-564.
 - [15] Sinha, Prashansa. "Cloud computing using AWS: an analysis." *Indian Journal of Computer Science* 5.6 (2020): 27-35.