International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

# API Mesh Architectures for Multi-Cloud Enterprises: A Comparative Analysis of Service Mesh, API Gateway, and Hybrid Governance Models

**Dr.M. Sunil Kumar**

Professor & Dean, Planning and Monitoring, School of Computing, Mohan Babu University, Tirupati, Andhra Pradesh, India

**Publication History**: Received: 04.09.2026; Revised: 02.10.2026; Accepted: 07.10.2026; Published: 20.10.2026.

**ABSTRACT:** Multi-cloud enterprises typically govern consumer-facing (north–south) traffic with API gateways and service-to-service (east–west) traffic with service meshes. Each pattern is mature within its own scope, but when they are deployed independently, enterprises inherit fragmented identity models, duplicated telemetry pipelines, inconsistent policy languages, and incomplete audit trails. This paper proposes an API Mesh reference architecture: a coordinated, vendor-agnostic integration fabric of seven layers—consumer access, API gateway, service mesh, workload identity, policy control, observability, and lifecycle governance—that share a common identity, policy-as-code, and telemetry substrate. Following Design Science Research Methodology (DSRM), the study defines the problem space, specifies the artifact, and evaluates gateway-only, mesh-only, and hybrid API Mesh patterns across ten dimensions: security, zero-trust alignment, latency, scalability, observability, governance, resilience, operational complexity, regulatory compliance, and cost. The architecture is demonstrated through four illustrative enterprise scenarios spanning global SaaS, financial services, manufacturing with IoT, and hybrid SAP landscapes, and a seven-phase migration roadmap is presented. The analysis indicates that a hybrid API Mesh provides broader governance coverage, end-to-end zero-trust enforcement, and fully correlated consumer-to-service audit trails, at the cost of higher operational complexity and modest additional latency. The paper closes with an agenda for empirical benchmarking and AI-assisted mesh operations.

**KEYWORDS:** API mesh, service mesh, API gateway, multi-cloud architecture, zero-trust architecture, policy-as-code, observability, enterprise integration governance, Kubernetes, design science research.

## I. INTRODUCTION

Cloud-native computing has restructured the enterprise integration landscape. Organizations that once relied on centralized enterprise service buses and perimeter-based security now operate distributed service estates spanning several public clouds, private data centers, Kubernetes clusters, serverless runtimes, legacy middleware, and partner networks. Recent work on event-driven integration has shown that asynchronous traffic in such estates benefits from a mesh-style fabric with shared routing, security, and governance across clouds [1]. Synchronous API traffic presents a parallel, and still largely unaddressed, governance challenge.

Two dominant patterns address different segments of this problem. API gateways [2]–[4] provide a managed entry point for consumer-facing traffic, centralizing authentication, rate limiting, schema validation, versioning, developer onboarding, and API product lifecycle management. Service meshes [5]–[7] provide a transparent data plane for service-to-service communication, adding mutual TLS (mTLS), traffic shaping, circuit breaking, retries, and workload-level policy enforcement without application code changes. The Kubernetes Gateway API [8] is a more recent effort to express mesh and gateway configuration through a common Kubernetes-native model. The two patterns are complementary rather than substitutive.

In multi-cloud enterprises neither pattern is sufficient on its own. Gateways deployed without mesh-layer governance become centralized bottlenecks, cannot enforce service-level zero trust [9], and cannot observe internal call chains.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

Meshes deployed without gateway-layer lifecycle management expose services without mature versioning, consumer analytics, or external access governance. When both exist without coordination, they produce fragmented identity models, duplicated observability pipelines, inconsistent policy languages, and operational complexity that outpaces most platform teams.

The concept of an API Mesh—an integration fabric that unifies gateway-level API governance with mesh-level service connectivity through shared identity, observability, and policy systems—has received limited formal treatment. This paper addresses that gap with a seven-layer reference architecture grounded in zero-trust principles [9], [10], Cloud Native Computing Foundation (CNCF) standards [7], [8], open policy frameworks [11], and established vendor-agnostic multi-cloud integration models [12].

A. Research Questions

**RQ1:** Which architectural responsibilities should be assigned to API gateways versus service meshes in multi-cloud enterprise environments?
**RQ2:** How does a hybrid API Mesh improve governance and observability relative to gateway-only or mesh-only deployments?
**RQ3:** What trade-offs arise across security, latency, operational complexity, and resilience among the three patterns?
**RQ4:** How can a seven-layer reference architecture support zero-trust adoption and policy-as-code governance at enterprise scale?
**RQ5:** What migration strategy allows an enterprise to evolve incrementally toward a hybrid API Mesh?

B. Contributions

This paper makes six contributions:
**(C1)** A formal separation of API gateway, service mesh, and API Mesh responsibilities expressed as a seven-layer, vendor-agnostic reference model.
**(C2)** A ten-dimension comparative evaluation framework for the three integration patterns, with supporting comparison tables.
**(C3)** A hybrid API Mesh reference architecture integrating zero-trust workload identity, policy-as-code, and correlated observability [13], [14].
**(C4)** Four illustrative enterprise scenarios—SaaS, financial services, manufacturing-IoT, and hybrid SAP—demonstrating the architecture's applicability.
**(C5)** A seven-phase incremental migration roadmap from fragmented integration estates to unified API Mesh governance.
**(C6)** A research agenda covering empirical benchmarking, AI-assisted mesh operations, and LLM-assisted policy authoring.

Section II reviews related work. Section III describes the methodology. Section IV presents the reference architecture, and Section V the comparative evaluation. Section VI demonstrates the architecture through enterprise scenarios, and Section VII presents the migration roadmap. Sections VIII–X discuss findings, threats to validity, and future research. Section XI concludes.

## II. RELATED WORK

A. API Gateways

API gateways emerged as the standard pattern for externalizing and governing service access. Fielding's REST constraints [15] established the architectural principles that gateways now enforce at runtime, and Masse [16] codified versioning, compatibility, and resource-design rules that gateways operationalize. The OWASP API Security Top 10 [17] identifies risks—broken object-level authorization, excessive data exposure, and unrestricted resource consumption among them—that gateways mitigate through policy enforcement. Commercial platforms such as Amazon API Gateway [2], Azure API Management [3], and Google Apigee X [4] add developer portals, monetization, and analytics. Enterprise case evidence indicates that gateway adoption at scale requires a structured modernization program covering governance, security, and CI/CD, rather than product configuration alone [18]. The principal limitation of gateway-centric architectures is that they govern consumer-to-service traffic only and cannot see or control the internal call graph of a microservice estate.

B. Service Meshes

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

Service mesh architectures were popularized by Linkerd [6] and Istio [5], the latter using the Envoy proxy [19] as its data plane. Zhu et al. [13] characterized sidecar overheads in detail and showed that mesh proxies can add substantial latency and CPU cost depending on configuration and protocol features, motivating sidecar-less and optimized data planes. The CNCF has advanced standardization through the Gateway API specification [8], including its GAMMA initiative for mesh traffic. Open problems in the mesh literature include control-plane operational burden, sidecar resource consumption, and immature multi-cluster and multi-cloud federation.

C. Zero-Trust Architecture

NIST SP 800-207 [9] formalizes the zero-trust model, in which no network location is implicitly trusted and access decisions are made per request based on identity, device posture, and context. Rose et al. articulate tenets that map directly onto API Mesh design: per-resource verification, identity-based and dynamic policy, authentication of every session, least privilege, and continuous monitoring. NIST SP 800-207A [10] extends this model to cloud-native, multi-location environments and identifies service mesh mTLS combined with gateway-layer authorization and an external policy engine such as Open Policy Agent (OPA) [11] as a practical realization of zero trust for microservices.

D. Multi-Cloud Integration and Networking

Multi-cloud integration research addresses consistent connectivity, security, and governance for services spread across AWS, GCP, Azure, Oracle Cloud, and private infrastructure. A vendor-agnostic integration model using standardized adapters and shared identity providers has demonstrated the feasibility of cross-cloud interoperability across these providers [12]. For asynchronous traffic, an event mesh reference framework has been proposed that federates brokers across clouds under shared governance and positions the event fabric as a foundation for AI-ready enterprises [1]. Burns et al. [20] traced the Borg–Omega–Kubernetes lineage and argued that platform-level abstraction is needed for workload portability across heterogeneous infrastructure. Multi-cloud network platforms such as Aviatrix [21] provide encrypted transit, segmentation, and routing governance across cloud boundaries. What remains underexplored is the interaction between network-layer connectivity and application-layer API governance for synchronous traffic—the gap an API Mesh addresses.

E. Policy-as-Code Governance

Policy-as-code encodes access control, compliance constraints, and operational rules in version-controlled, machine-evaluable form. OPA [11] has become the de facto policy engine in cloud-native environments, allowing teams to express authorization in Rego and evaluate it at gateway, mesh, and application layers. Declarative, pipeline-managed gateway policies have been reported to reduce configuration drift and improve auditability compared with console-driven administration [18]. HashiCorp Consul [22] offers comparable intention-based authorization within its own mesh. The policy-as-code literature rarely addresses synchronization of policy across gateway and mesh layers, leaving a risk of conflicting enforcement when one request traverses both.

F. Cloud-Native Observability

Majors et al. [23] define observability as the ability to understand a system's internal state from its external outputs, which requires structured logs, metrics, and distributed traces. OpenTelemetry [14] standardizes instrumentation and trace-context propagation, allowing a single request to be correlated across gateway, mesh proxy, and application. Site reliability engineering practice [24] contributes service level objectives, error budgets, and golden-signal monitoring. Mutyam [25] proposed graph-based modeling of service dependencies to predict failure propagation, showing that telemetry-derived service graphs can expose high-risk dependencies before incidents occur. Cross-layer correlation—linking gateway-level consumer context to mesh-level dependency traces—remains insufficiently treated.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

**TABLE I** Comparison of Representative Prior Studies

| Study | Focus | Approach | Contribution | Limitation |
|---|---|---|---|---|
| Rose et al. [9] | Zero-trust architecture | NIST SP 800-207 framework | Zero-trust tenets and logical components | No gateway–mesh coordination model |
| Chandramouli [10] | Zero trust for cloud-native apps | NIST SP 800-207A | Multi-location cloud-native ZTA model | Limited legacy-system guidance |
| Zhu et al. [13] | Service mesh overhead | Empirical characterization of sidecars | Quantified latency and CPU cost | Mesh layer only; no gateway context |
| Venkiteela [12] | Multi-cloud integration | Vendor-agnostic Boomi / SAP BTP model | Cross-cloud interoperability | No service mesh layer |
| Venkiteela [1] | Event-driven integration | Event mesh reference framework | Federated, governed async fabric | Asynchronous traffic only |
| Venkiteela [18] | API modernization | Apigee X enterprise program | Gateway governance and policy-as-code | No mesh-layer policy synchronization |
| Burns et al. [20] | Platform abstraction | Borg → Kubernetes lineage | Workload portability model | API governance not addressed |
| Mutyam [25] | Failure propagation | Graph-based dependency modeling | Service-graph risk prediction | Cross-layer telemetry gap |

Taken together, these studies reveal a consistent gap: no existing work formally specifies the coordinated design of gateway and mesh layers as one API Mesh with shared zero-trust identity, policy-as-code, and correlated observability across clouds. The present study addresses this gap.

## III. RESEARCH METHODOLOGY

This research follows Design Science Research (DSR), which produces and evaluates prescriptive artifacts—architectures, frameworks, or models—that address important organizational problems [26]. The study applies the six-activity Design Science Research Methodology (DSRM) process of Peffers et al. [27], and uses the Zachman framework [28] to keep architectural views consistent.

**Activity 1 – Problem identification:** the absence of a formal, vendor-agnostic model that coordinates gateway [2]–[4] and service mesh [5], [6] capabilities in multi-cloud enterprises, established through the review in Section II.

**Activity 2 – Objectives of a solution:** a reference architecture that delivers consistent workload identity [29], correlated observability [14], policy-as-code [11], zero-trust enforcement [9], [10], and incremental migration across heterogeneous clouds.

**Activity 3 – Design and development:** the seven-layer API Mesh architecture in Section IV, grounded in CNCF standards [7], [8], NIST guidance [9], [10], and OWASP API security [17].

**Activity 4 – Demonstration:** four illustrative enterprise scenarios (Section VI), each representing a distinct deployment context.

**Activity 5 – Evaluation:** an analytical, ten-dimension comparison of three patterns (Section V), with explicit threats to validity (Section IX).

**Activity 6 – Communication:** this paper, including comparison tables, architecture figures, and a migration roadmap.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

DSR is preferred over a purely empirical design because the API Mesh is an emerging construct for which no standardized benchmarking environment yet exists. Analytical and scenario-based evaluation are recognized DSR evaluation methods [26]; Section X identifies the controlled experiments needed to complement them.

## IV. API MESH REFERENCE ARCHITECTURE

The API Mesh is organized as seven layers, each with distinct responsibilities, inputs, outputs, security controls, and governance controls (Table II). The layers operate across heterogeneous clouds without mandating a single vendor. Fig. 1 shows the overall topology, including the trust boundary between the external (L1–L2) and internal (L3–L7) zones.

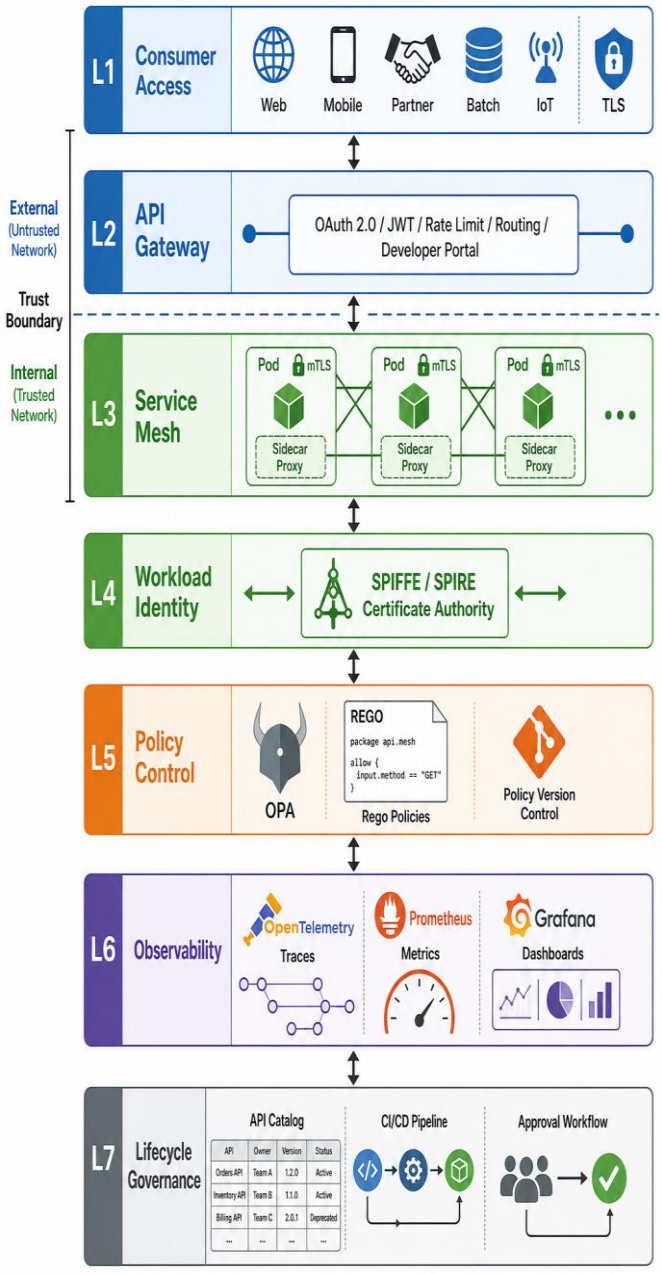



**Fig.1.** Seven-layer API Mesh reference architecture.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

**TABLE II** Seven-Layer API Mesh Reference Architecture

| Layer | Name and responsibilities | Inputs | Outputs | Security controls | Governance controls |
|---|---|---|---|---|---|
| L1 | Consumer access: TLS termination, WAF, DDoS protection, geo-routing | External requests | Filtered sessions | TLS 1.3, WAF, DDoS | Consumer registration |
| L2 | API gateway [2]–[4]: OAuth 2.0, JWT, rate limiting, routing, versioning, developer portal | Consumer tokens | Routed requests | OAuth 2.0, JWT, quotas | API lifecycle, SLAs |
| L3 | Service mesh [5], [6]: mTLS, traffic splitting, retries, circuit breaking, telemetry | Internal service requests | Governed responses | mTLS, SPIFFE identities [29] | Traffic policy, canary release |
| L4 | Workload identity [29]: SPIFFE/SPIRE certificate issuance, federated trust, service-account mapping | Deployment metadata | X.509 SVIDs | Short-lived certificates, rotation | Identity federation, trust anchors |
| L5 | Policy control [11]: Rego rules, RBAC/ABAC, data residency, compliance constraints | Identity and request context | Allow/deny decisions | Least privilege | Version-controlled, peer-reviewed policy |
| L6 | Observability [14], [24]: tracing, metrics, structured logs, dependency graphs | Telemetry from L1–L5 | Correlated traces, SLO dashboards | Anomaly detection, alerting | SLO tracking, incident correlation |
| L7 | Lifecycle governance: API catalog, deployment approvals, security review, operational readiness | Artifacts, audit logs | Approved releases, audit records | SAST/DAST gates, vulnerability scanning | Catalog, change approval, compliance |

A. Layer Design Principles

Each layer operates independently but shares primitives with its neighbors. The separation between consumer identity (OAuth tokens at L2) and workload identity (SPIFFE SVIDs at L4) is architecturally critical: the token proves consumer context at the gateway, while the SVID proves service identity inside the mesh [9], [29]. Mapping between the two identity domains is governed by explicit trust policy at L5 rather than by implicit credential propagation. The policy layer also implements separation of duties—security teams own baseline encryption and authentication rules, platform teams own routing and availability policy, and compliance teams own data-handling constraints [11]—with all three evaluated by a single engine so that gateway and mesh decisions cannot diverge. The observability layer [14], [24], [25] joins telemetry from every other layer into a single request journey, allowing root-cause analysis across gateway, mesh, policy, and application boundaries.

B. Relationship to the Event Mesh

The API Mesh governs synchronous REST, gRPC, and GraphQL traffic. Its asynchronous counterpart is the event mesh, which federates brokers and event streams across clouds [1]. The two fabrics can share L4–L7: a single workload-identity domain, one policy engine, one telemetry pipeline, and one catalog covering both APIs and event channels. Section X treats this convergence as a direction for future work.

## V. COMPARATIVE EVALUATION

This section evaluates API Gateway only (GW), Service Mesh only (SM), and Hybrid API Mesh (AM) across ten dimensions. Three dimensions—security, latency, and governance—are split into two sub-dimensions each, giving thirteen rows in Table III. Each row is rated on an ordinal scale (Limited, Moderate, Strong; complexity and cost use

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

Low–High). Ratings are derived analytically from the literature in Section II and the layer responsibilities in Section IV, and are illustrated by the scenarios in Section VI. Fig. 2 visualizes the comparison.

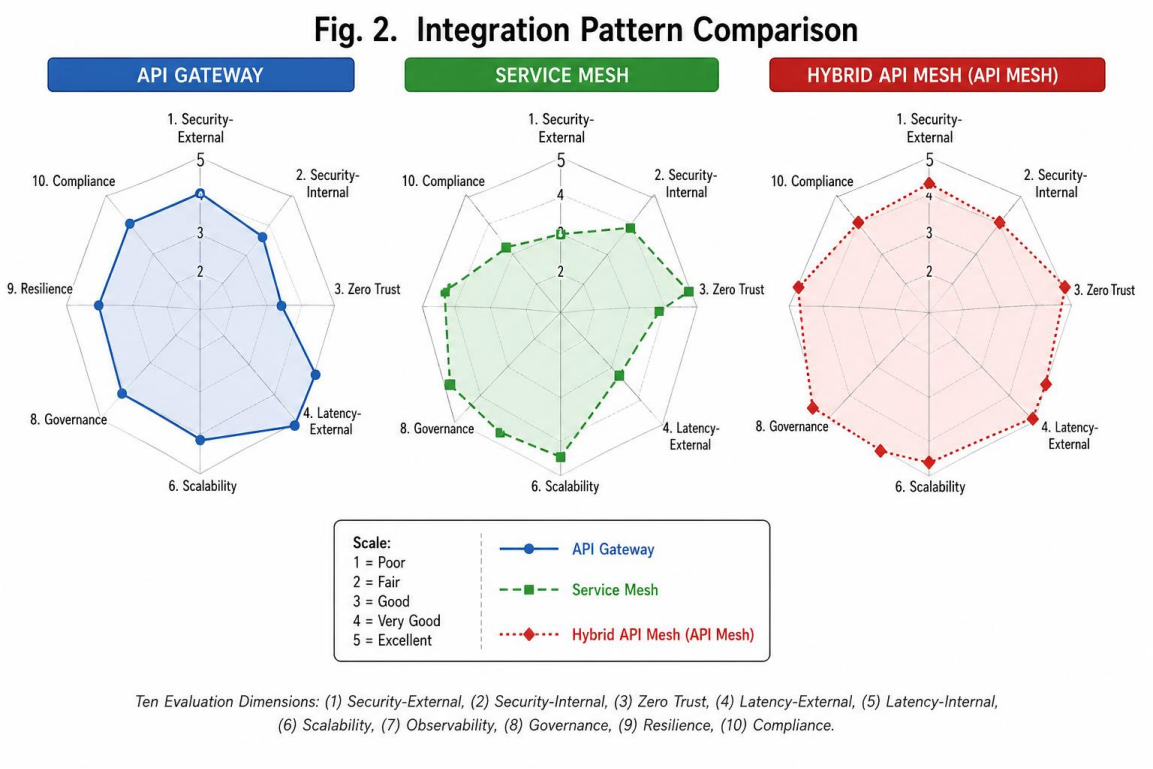



**Fig.2.** Integration-pattern comparison across the evaluation dimensions.

**TABLE III** Comparative Evaluation of Integration Patterns

| Dimension | API gateway only | Service mesh only | Hybrid API Mesh |
|---|---|---|---|
| 1a. Security – external API [17] | Strong | Limited | Strong |
| 1b. Security – service-to-service [9] | Limited | Strong | Strong |
| 2. Zero-trust alignment [10] | Moderate | Moderate | Strong |
| 3a. Latency – external path [13] | Moderate | Strong | Moderate |
| 3b. Latency – internal path [13] | Limited | Moderate | Moderate |
| 4. Scalability | Moderate | Strong | Strong |
| 5. Observability [14], [25] | Moderate | Moderate | Strong |
| 6a. Governance – API lifecycle | Strong | Limited | Strong |
| 6b. Governance – service traffic | Limited | Strong | Strong |
| 7. Resilience | Limited | Strong | Strong |
| 8. Operational complexity (lower is better) | Moderate | Moderate | High |
| 9. Regulatory compliance [10], [11] | Moderate | Limited | Strong |
| 10. Cost – initial adoption (lower is better) | Low–Moderate | Moderate | Moderate–High |

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

The hybrid API Mesh is rated Strong on security, zero-trust alignment, scalability, observability, governance, resilience, and compliance—seven of the ten dimensions. It is rated Moderate on latency, because external requests traverse both a gateway and a mesh proxy [13], and it carries the highest operational complexity and initial cost. These trade-offs are consistent with the general observation that broader architectural coverage demands greater platform-engineering investment; they are best managed through automation and clear ownership boundaries rather than by restricting the architecture.

**TABLE IV** Security Sub-Analysis

| Security control | API gateway | Service mesh | API Mesh |
|---|---|---|---|
| OAuth 2.0 / JWT validation | Native | Limited | Native (L2) |
| Mutual TLS [5], [6] | Absent | Native | Native (L3) |
| SPIFFE/SPIRE workload identity [29] | Absent | Supported | Native (L4) |
| Schema / input validation [17] | Native | Absent | Native (L2) |
| Rate limiting / throttling | Native | Limited | Native (L2) |
| Policy-as-code (OPA) [11] | Extensible | Extensible | Native (L5) |
| End-to-end zero trust [9], [10] | Partial | Partial | Complete |
| Audit trail (consumer → service) | Consumer-level only | Service-level only | Fully correlated |

## VI. MULTI-CLOUD SCENARIO DEMONSTRATION

Four illustrative scenarios demonstrate how the architecture applies in distinct multi-cloud contexts. Each is a composite drawn from common enterprise deployment patterns. The outcomes listed are indicative design-level expectations, not results of controlled measurement; they are reported to show the kind of benefit each layer is intended to deliver and should be validated empirically (Section X). Fig. 3 shows the multi-cloud topology shared by the scenarios.

Fig. 3. Multi-Cloud API Mesh Topology

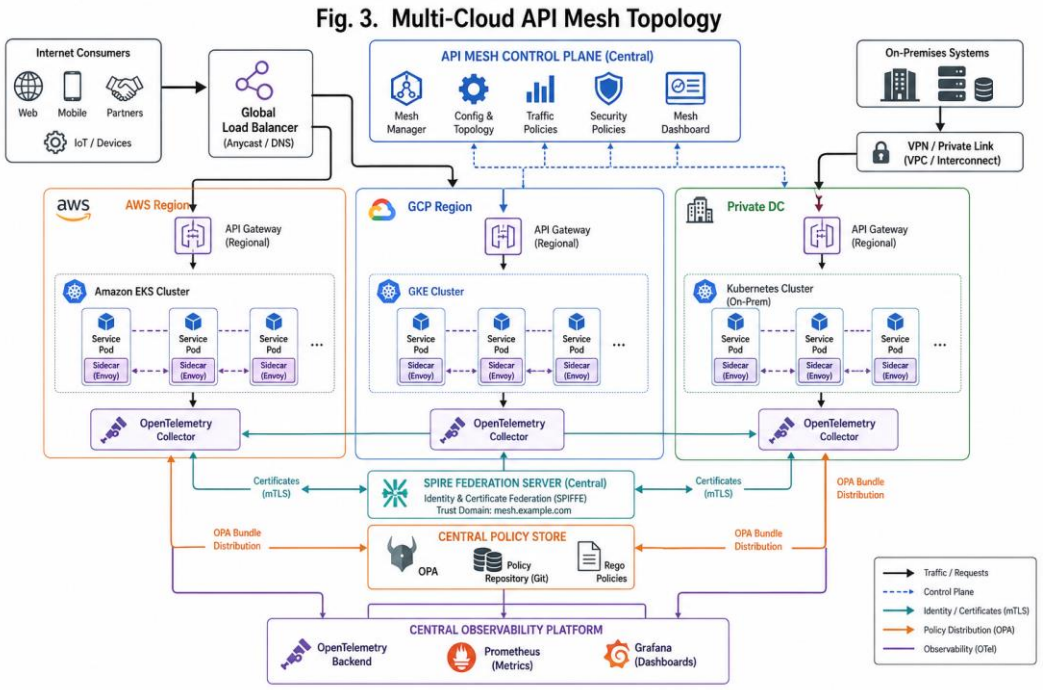



**Fig.3.** Multi-cloud API Mesh topology across AWS, GCP, and a private data center

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

A. Scenario 1 – Global SaaS Platform
A SaaS provider runs in three regions across AWS and GCP, serving several hundred enterprise tenants through OAuth-secured REST and GraphQL APIs at tens of millions of calls per day. Challenges include consistent cross-region latency, per-tenant rate limiting, observability across more than one hundred microservices, and zero-trust enforcement [9] for multi-tenant workloads.

API Mesh solution. Apigee X [4] provides tenant isolation, quotas, and versioning at L2; Istio [5] governs service-to-service traffic in both clouds at L3; SPIFFE/SPIRE [29] federates workload identity across regions at L4; OPA [11] enforces tenant-scoped data isolation at L5; and OpenTelemetry [14] supplies correlated consumer-to-service traces at L6. Indicative outcomes: faster detection of cross-service incidents through correlated traces, uniform mTLS without application changes, and self-service API onboarding through the developer portal.

B. Scenario 2 – Financial Services Enterprise
A regulated institution runs core banking on private infrastructure, digital banking APIs on Azure [3], and analytics on AWS [2]. Requirements include GDPR data residency, PCI-DSS segmentation, SOX audit completeness, and continuous control monitoring; shadow-API proliferation is the main governance risk.

API Mesh solution. Azure API Management [3] governs digital banking APIs with OAuth 2.0 and certificate-based B2B authentication. Consul [22] provides an mTLS mesh across on-premises and Azure workloads. A central OPA [11] policy service enforces data-residency rules on every route, and OpenTelemetry [14] with compliance-grade retention supports SOX audit. An L7 catalog records every endpoint with its owner, consumers, and compliance classification. Indicative outcomes: a complete audit trail for regulated calls, policy-enforced data residency, shorter security review cycles through standardized controls, and discovery of undocumented APIs through gateway traffic analysis.

C. Scenario 3 – Manufacturing and IoT
A manufacturer operates thousands of IoT devices across dozens of plants, ingesting telemetry through AWS IoT services, processing it in GCP, and integrating with SAP ERP for production scheduling. Edge sites run lightweight Kubernetes (k3s). The challenge is consistent security and observability from device to ERP across a fully heterogeneous estate.

API Mesh solution. Amazon API Gateway [2] handles device registration and ingestion with certificate-based authentication; Linkerd [6] governs processing microservices in GCP; SPIFFE SVIDs [29] federate identity across edge and cloud; a single OPA bundle [11] enforces data classification across all clusters; and Prometheus with OpenTelemetry [14] provides edge-to-cloud observability. Indicative outcomes: earlier anomaly detection, mTLS between edge and cloud services without device-firmware changes, and uniform policy across clusters.

D. Scenario 4 – Hybrid SAP Landscape
A technology enterprise runs SAP S/4HANA on private infrastructure, SAP BTP Integration Suite for cloud integration, Salesforce for CRM, Workday for HR, and SAP Ariba for procurement, with several hundred integration flows across AWS, GCP, and on-premises systems. Legacy Oracle SOA middleware still carries some financial integrations.

API Mesh solution. SAP BTP API Management governs SAP-facing traffic with SAP-native OAuth, SAML, and service-binding authentication, while Apigee X [4] governs non-SAP traffic behind a unified developer portal—an application of the vendor-agnostic model in [12]. A shared OpenTelemetry [14] collector correlates telemetry across both gateways. OPA [11] enforces procurement approvals, journal segregation, and HR data access through one policy bundle. Legacy Oracle SOA flows are placed behind the gateway as managed endpoints, so they can be migrated to BTP without disrupting consumers. Indicative outcomes: lower order-processing latency after migration, a single catalog covering all integration APIs, one policy set across platforms, and retirement of legacy middleware without consumer-visible disruption.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

Fig. 4. Policy and Identity Flow

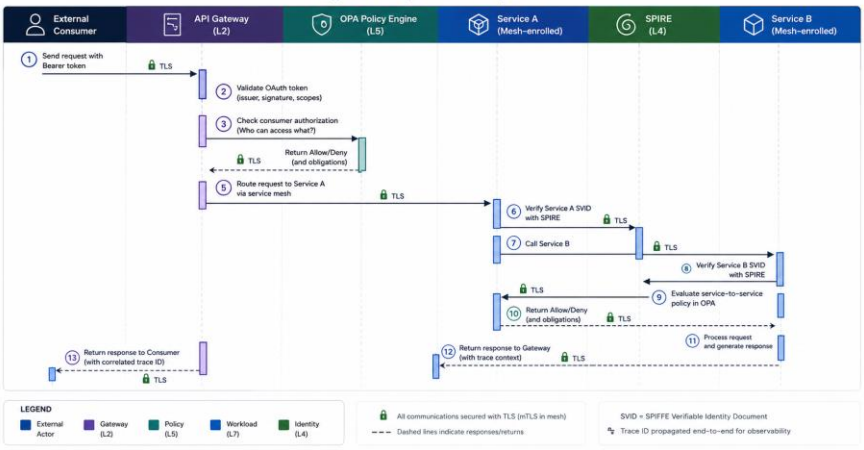



**Fig.4.** Policy and identity flow across API Mesh layers.

## VII. MIGRATION ROADMAP

Few enterprises can adopt all seven layers at once. Fig. 5 presents a seven-phase, approximately 24-month roadmap in which phases overlap and each ends at a completion gate that confirms readiness for the next.

**Phase 1 – API inventory and shadow-API discovery (months 1–2):** catalog all APIs and integration flows, identify owners, and use gateway and network telemetry to find undocumented endpoints.

**Phase 2 – Gateway rationalization (months 2–4):** consolidate overlapping gateways, standardize policy templates, and route all external traffic through governed L2 entry points.

**Phase 3 – Mesh adoption for priority domains (months 3–6):** introduce the mesh in the domains with the most complex dependency graphs or strictest regulatory needs.

**Phase 4 – Workload identity integration (months 5–8):** deploy SPIFFE/SPIRE [29], federate trust across clusters and clouds, and map consumer identities to workload identities at L5.

**Phase 5 – Observability correlation (months 7–10):** standardize on OpenTelemetry [14] across gateways, proxies, and applications, and define SLOs for critical journeys.

**Phase 6 – Policy-as-code rollout (months 9–12):** migrate gateway and mesh authorization rules into a single, version-controlled OPA [11] repository with CI-based testing.

**Phase 7 – Lifecycle governance and continuous improvement (months 12–24):** integrate the catalog, approval workflows, and security gates into delivery pipelines, and review the architecture against SLOs and audit findings.

***Fig.* 5. API Mesh Migration Roadmap**

*24-Month Phased Roadmap from Planning to Operational Excellence*




**Fig.5.** Seven-phase API Mesh migration roadmap (24-month plan).

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

## VIII. DISCUSSION

A. Principal Findings

In answer to RQ1 and RQ2, gateways and meshes solve structurally distinct but complementary problems, and treating them as alternatives creates deficits in both directions. The hybrid API Mesh separates concerns by layer while sharing identity [29], observability [14], and policy [11] infrastructure. In answer to RQ3, the hybrid pattern trades additional latency and operational complexity for broader security, governance, and audit coverage.

B. Multi-Cloud Governance Implications

Multi-cloud environments make architectural consistency more important than tool selection. Governance quality depends less on which products are chosen than on whether identity, telemetry, policy, and lifecycle controls are consistent enough across them to support security operations, incident response, and audit [12]. This suggests that interface standardization—requiring every gateway to emit OpenTelemetry traces, every mesh to accept SPIFFE identities [29], and every authorization decision to be evaluated by OPA [11]—is more effective than mandating a single vendor. This addresses RQ4.

C. Reliability Through Correlated Telemetry

Correlated API Mesh telemetry can improve reliability beyond what gateway or mesh telemetry provides separately. The dependency graphs derivable from correlated traces [14] provide exactly the structure that failure-propagation models such as Mutyam's [25] require, enabling earlier detection of failure patterns, identification of high-risk dependencies, and linkage of business impact to technical root cause. This extends SRE practice [24] into the multi-cloud API context.

D. Adoption Barriers

Three barriers stand out. First, operational complexity: gateway and mesh configuration, certificate rotation, policy engines, and telemetry pipelines demand more capacity than many platform teams have. Second, organizational alignment: security, platform, application, and compliance teams must agree on ownership before implementation. Third, legacy integration: workloads that cannot run mesh proxies fall outside mesh governance. The architecture addresses these through its governance layer, shared identity model [29], gateway-based abstraction of legacy endpoints, and the phased roadmap in Section VII, which answers RQ5.

## VIII. THREATS TO VALIDITY

A. Internal Validity

The ratings in Section V come from architectural analysis and literature synthesis rather than controlled measurement. This is mitigated by grounding ratings in cited sources [9]–[11], [13], [14], [25], applying one rubric consistently to all three patterns, and illustrating each rating through the scenarios.

B. External Validity

The four scenarios are composites and may not represent every deployment context. Domains such as defense, healthcare, and energy may impose constraints not captured here. Controlled benchmarks in these domains are needed to establish the limits of generalizability.

C. Construct Validity

The reference architecture is prescriptive rather than a description of any single deployed system, and mapping implementation choices to layers involves interpretation. Table II mitigates this by defining explicit inputs, outputs, and controls for each layer, consistent with the Zachman framework [28].

D. Conclusion Validity

Because the evaluation framework and the API Mesh were designed together, there is a risk of circularity in concluding that the hybrid pattern offers broader coverage. This is partly mitigated by independent evidence that neither gateway-only [2], [3] nor mesh-only [5], [6] deployments meet the full requirement set in the zero-trust literature [9], [10]. Independent empirical evaluation remains necessary.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

## IX. FUTURE RESEARCH

Empirical benchmarking. The most immediate need is controlled measurement of gateway-only, mesh-only, and hybrid configurations on AWS, GCP, and Azure, covering request latency, policy-propagation time, certificate-rotation impact, telemetry overhead [13], and developer productivity.

AI-assisted mesh operations. Models trained on API Mesh telemetry—gateway logs, mesh traces, dependency graphs, and policy decisions—could support real-time anomaly detection, failure-propagation prediction [25], identification of unused APIs, and capacity forecasting.

LLM-assisted policy authoring. Large language models could translate natural-language compliance requirements into OPA [11] Rego policies. This requires formal verification of generated policy, controls for hallucination in security-critical contexts, and alignment between intent and machine-evaluable semantics.

Retrieval-augmented governance. Retrieval-augmented generation [30] could allow natural-language querying of policy catalogs, API inventories, and telemetry stores, while emerging agent-interoperability protocols such as the Model Context Protocol [31] raise the question of how API Mesh controls should govern traffic generated by AI agents.

Event mesh convergence. Enterprises increasingly combine synchronous APIs with event streams. Extending the architecture so that one identity, policy, and observability substrate governs both the API Mesh and the event mesh [1] is a significant research frontier.

Platform engineering. Future work should examine how API Mesh capabilities can be packaged as internal developer platform services, letting application teams provision governed API products and mesh policies through self-service workflows.

## X. CONCLUSION

Multi-cloud enterprises need an integration architecture that is secure, observable, resilient, and governable across heterogeneous platforms and organizational boundaries. This paper has argued that neither API gateways [2]–[4] nor service meshes [5], [6] meet this need alone, and has proposed a seven-layer hybrid API Mesh that coordinates gateway-level API governance with mesh-level service connectivity through shared workload identity [29], policy-as-code [11], and correlated observability [14].

The comparative evaluation rates the hybrid API Mesh Strong on seven of ten dimensions, with the trade-offs concentrated in latency, operational complexity, and initial cost. Four illustrative scenarios show its applicability to SaaS, regulated financial services, industrial IoT, and hybrid SAP contexts, and the seven-phase roadmap offers an incremental adoption path. Together with the event mesh framework for asynchronous traffic [1], the API Mesh provides a foundation for consistently governed, vendor-agnostic integration in multi-cloud enterprises. Empirical benchmarking is the most important next step.

**DECLARATIONS**

Funding: This research received no specific grant from any funding agency.

Conflict of interest: The author declares no conflict of interest. Product names are used for illustration only; no vendor sponsored or reviewed this work.

Data availability: No datasets were generated or analyzed in this study.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

## REFERENCES

[1] P. Venkiteela, "Enterprise integration architecture using event mesh: A reference framework for cloud-native and AI-ready enterprises," Methods in Science and Technology Studies, vol. 2, no. 2, pp. 208–223, 2026, doi: 10.64539/msts.v2i2.2026.559.
[2] Amazon Web Services, "Amazon API Gateway developer guide," 2026. [Online]. Available: https://docs.aws.amazon.com/apigateway/
[3] Microsoft, "Azure API Management documentation," 2026. [Online]. Available: https://learn.microsoft.com/en-us/azure/api-management/
[4] Google Cloud, "Apigee documentation," 2026. [Online]. Available: https://cloud.google.com/apigee/docs
[5] Istio Authors, "Istio documentation," 2026. [Online]. Available: https://istio.io/latest/docs/
[6] Linkerd Authors, "Linkerd documentation," 2026. [Online]. Available: https://linkerd.io/
[7] Cloud Native Computing Foundation, "CNCF cloud native landscape," 2026. [Online]. Available: https://landscape.cncf.io/
[8] Kubernetes SIG Network, "Gateway API," 2026. [Online]. Available: https://gateway-api.sigs.k8s.io/
[9] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," NIST, Gaithersburg, MD, USA, Special Publication 800-207, Aug. 2020, doi: 10.6028/NIST.SP.800-207.
[10] R. Chandramouli, "A zero trust architecture model for access control in cloud-native applications in multi-location environments," NIST, Gaithersburg, MD, USA, Special Publication 800-207A, Sep. 2023, doi: 10.6028/NIST.SP.800-207A.
[11] Open Policy Agent Authors, "Open Policy Agent documentation," 2026. [Online]. Available: https://www.openpolicyagent.org/docs
[12] P. Venkiteela, "Vendor-agnostic integration across AWS/GCP/OCI using Boomi and SAP BTP," Journal of Engineering Research and Sciences, 2025, doi: 10.55708/js0412001.
[13] X. Zhu et al., "Dissecting overheads of service mesh sidecars," in Proc. ACM Symp. Cloud Computing (SoCC), 2023, doi: 10.1145/3620678.3624652.
[14] OpenTelemetry Authors, "OpenTelemetry documentation," 2026. [Online]. Available: https://opentelemetry.io/docs/
[15] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.
[16] M. Masse, REST API Design Rulebook. Sebastopol, CA, USA: O'Reilly Media, 2011.
[17] OWASP Foundation, "OWASP API Security Top 10 – 2023," 2023. [Online]. Available: https://owasp.org/API-Security/
[18] P. Venkiteela, "Strategic API modernization using Apigee X for enterprise transformation," Journal of Information Systems Engineering and Management, 2024. [Online]. Available: https://www.jisem-journal.com/index.php/journal/article/view/13168
[19] Envoy Project Authors, "Envoy proxy documentation," 2026. [Online]. Available: https://www.envoyproxy.io/docs
[20] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Communications of the ACM, vol. 59, no. 5, pp. 50–57, May 2016, doi: 10.1145/2890784.
[21] Aviatrix Systems, "Aviatrix documentation," 2026. [Online]. Available: https://docs.aviatrix.com/
[22] HashiCorp, "Consul service mesh documentation," 2026. [Online]. Available: https://developer.hashicorp.com/consul/docs
[23] C. Majors, L. Fong-Jones, and G. Miranda, Observability Engineering. Sebastopol, CA, USA: O'Reilly Media, 2022.
[24] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA, USA: O'Reilly Media, 2016.
[25] N. Mutyam, "Graph-based modeling of service dependencies for predicting failure propagation in distributed systems," International Journal of Multidisciplinary Evolutionary Research, vol. 5, no. 1, pp. 113–116, 2024, doi: 10.54660/IJMER.2024.5.1.113-116.
[26] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," MIS Quarterly, vol. 28, no. 1, pp. 75–105, Mar. 2004, doi: 10.2307/25148625.
[27] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," Journal of Management Information Systems, vol. 24, no. 3, pp. 45–77, 2007, doi: 10.2753/MIS0742-1222240302.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 5, September-October 2026||

DOI:10.15662/IJRPETM.2026.0905006

[28] J. A. Zachman, "A framework for information systems architecture," IBM Systems Journal, vol. 26, no. 3, pp. 276–292, 1987, doi: 10.1147/sj.263.0276.

[29] SPIFFE Project, "SPIFFE: Secure Production Identity Framework for Everyone," 2026. [Online]. Available: https://spiffe.io/docs/

[30] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459–9474.

[31] Model Context Protocol, "Model Context Protocol specification," 2025. [Online]. Available: https://modelcontextprotocol.io/specification