www.ijbar.org
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Secure API Gateways in Multi-Cloud Architectures: Performance and Policy Enforcement

Venkata Jyothi Swaroop Chejarla

Independent Researcher, India

Abstract

As enterprise architectures transition to microservices and containerized applications, the use of multiple cloud providers has become a strategic imperative for agility and availability. However, this multi-cloud landscape introduces significant challenges in securing and managing API traffic that spans disparate platforms. This paper evaluates the deployment and performance of secure API gateways across multi-cloud environments, focusing on three widely adopted platforms: Kong, AWS API Gateway, and Apigee. The analysis covers authentication mechanisms, rate limiting, input validation, JWT-based access control, and the enforcement of OpenAPI-defined policies. A performance simulation involving 10,000 concurrent API requests across AWS and GCP regions reveals that Kong and Apigee deliver more consistent response times under load, while AWS API Gateway excels in granular IAM-based access control. Misconfigurations—including unauthenticated routes, overly permissive policies, and lack of input validation—were found to cause broken authentication and internal API exposure. To address these issues, we propose a standardized API policy enforcement framework and advocate centralized logging through cloud-native SIEM tools like AWS CloudTrail and Google Cloud Logging. The study concludes that robust and uniform API security is not just a best practice but a necessity in multi-cloud microservice ecosystems, and it must be embedded into cloud governance strategies from the outset.

Keywords: API gateway, multi-cloud, Kong, AWS API Gateway, Apigee, JWT, OpenAPI, rate limiting, input validation, SIEM integration, cloud governance

---

1. Introduction

With the proliferation of microservices, DevOps pipelines, and hybrid deployments, APIs have become the de facto interface for enterprise functionality. APIs enable interoperability, modularity, and rapid iteration—but they also expose organizations to attack surfaces that are difficult to monitor, govern, and secure across distributed cloud environments. The stakes are particularly high in **multi-cloud architectures**, where API traffic flows across heterogeneous infrastructures such as AWS, Google Cloud Platform (GCP), Azure, and on-premises environments.

API gateways act as the **control plane** for API traffic. They provide central capabilities such as rate limiting, access control, input sanitation, authentication, and threat prevention. In a multi-cloud context, the challenge is not only to implement these capabilities but to do so **consistently and securely** across vendors—without introducing latency, policy drift, or management overhead.

This paper investigates the security posture and performance characteristics of three leading API gateway platforms:

- **Kong Gateway** (open-source and enterprise)
- **AWS API Gateway** (native to the AWS ecosystem)

- **Apigee** (managed API gateway solution from Google Cloud)

We explore the extent to which these platforms support:

- **Standards-based policy enforcement**, such as OpenAPI 3.0 specifications and JWT-based authentication
- **Advanced rate limiting and threat protection**, including protection against common API-level attacks (e.g., SQL injection, XSS)
- **Policy consistency and logging** across cloud environments

The paper is organized as follows: Section 2 reviews related work and prior evaluations of API security in distributed systems. Section 3 defines the hypotheses driving our evaluation. Section 4 details our methodology, including deployment topology, traffic simulation, and performance measurement. Section 5 presents the results. Section 6 offers a security policy framework and analysis of common misconfigurations. Section 7 concludes with recommendations for integrating secure API governance into multi-cloud strategies.

---

2. Literature Review

2.1 Rise of Multi-Cloud and API Proliferation

As enterprises adopt a **multi-cloud strategy** to avoid vendor lock-in and optimize for regional availability, API traffic increasingly traverses multiple clouds. According to Flexera's 2021 State of the Cloud Report, over 92% of enterprises use more than one public cloud provider. This introduces operational silos, making **centralized security enforcement difficult**, particularly when using native gateways provided by different vendors.

2.2 API Gateways and Zero Trust Architectures

API gateways are critical to **Zero Trust security models**. They serve as enforcement points for least privilege access, token validation, and anomaly detection. Recent research (e.g., Alshamrani et al., 2020) has emphasized the importance of integrating gateways into security orchestration pipelines. However, **most existing evaluations focus on single-cloud performance**, leaving a gap in understanding how gateways behave in federated and cross-cloud topologies.

2.3 Security Features: JWT, OpenAPI, and Threat Protection

Industry best practices (OWASP API Top 10, 2019) recommend that APIs implement:

- **Authentication** using secure tokens (e.g., OAuth2 with JWT)
- **Input validation** to prevent injection attacks
- **Rate limiting and quotas** to mitigate abuse
- **Schema enforcement** using OpenAPI specifications

Some vendors provide out-of-the-box support for these features, while others require additional configuration or external middleware. The effectiveness of these controls in real-world multi-cloud scenarios remains under-explored.

2.4 Logging and Observability

www.ijbar.org
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Logging plays a crucial role in detecting misconfigurations and abuse. Cloud-native SIEM solutions like **AWS CloudTrail**, **GCP Cloud Logging**, and **Azure Monitor** are capable of ingesting API gateway logs, but **policy fragmentation** and **format discrepancies** often hinder correlation across platforms.

Our work builds on prior research by evaluating how well these gateways can **enforce uniform security policies and scale under load** in a federated environment.

---

3. Hypotheses

This study is guided by the following hypotheses, formulated to assess both the **security capabilities** and **performance efficiency** of API gateways across multi-cloud environments:

- **H1**: Secure API gateways can enforce consistent authentication, rate limiting, and input validation policies across cloud providers using OpenAPI and JWT standards.
- **H2**: Under concurrent request loads (≥10,000), Kong and Apigee will demonstrate lower latency variance and more stable throughput than AWS API Gateway.
- **H3**: Misconfigured gateways will result in common security failures, including broken authentication and overexposure of internal APIs.
- **H4**: Centralized logging through cloud-native SIEM tools enables effective correlation and auditing of API gateway events across AWS and GCP platforms.

These hypotheses form the basis of our performance benchmarking and misconfiguration analysis in a federated cloud deployment.

---

4. Methodology

This section describes the experimental setup for evaluating the selected API gateways in a federated, production-simulated multi-cloud architecture.

4.1 Platforms Evaluated

The study evaluates three API gateway platforms:

- **Kong Gateway v3.0 (Open Source)**, deployed in containers on AWS EC2 and GCP Compute Engine.
- **AWS API Gateway (REST mode)**, configured with Lambda backends and IAM authorizers.
- **Apigee Edge**, deployed via Google Cloud's managed interface, connected to Cloud Run and Cloud Functions.

Each gateway was configured to handle:

- OpenAPI 3.0-compliant routes
- OAuth2/JWT authentication
- Input validation schemas
- IP-based rate limiting (per minute)

4.2 Deployment Topology

Index in Cosmos
MAY 2025, Volume 15, ISSUE 2
UGC Approved Journal

The testbed was deployed across AWS (us-east-1) and GCP (us-central1):

- Each gateway exposed identical APIs (GET, POST, PUT endpoints).
- Backends returned simulated payloads (1 KB and 5 KB).
- Cloud load balancers were used to distribute traffic.
- Gateways were instrumented to log to **CloudTrail** (AWS), **GCP Logging**, and **Prometheus-Grafana** (for Kong).

Figure 1 (to be included) will illustrate the architecture and traffic flow between clients, gateways, and backend services.

4.3 Security Test Cases

To assess security enforcement and misconfiguration resilience, we conducted the following tests:

Test Type	Description
JWT Spoofing	Attempts to bypass route authorization using tampered JWT tokens
Policy Drift	Gateway evaluated for inconsistencies between documented and active rules
Input Injection	Payloads crafted for SQLi and XSS pattern injection
Open Route Discovery	Port scans and unauthorized access attempts to internal services

Each test was run with baseline configurations and after hardening based on vendor recommendations.

4.4 Performance Benchmarking

API load testing was performed using **k6** and **Artillery.io**, simulating:

- **10,000 concurrent requests** (5k AWS, 5k GCP origin)
- Request rates of 500 to 2,000 requests per second
- Both authenticated and unauthenticated traffic flows

Metrics recorded included:

- **Average response time**
- **95th percentile latency**
- **Throughput (requests/sec)**
- **Error rate (%)**
- **CPU/memory utilization for Kong container deployments**

Each test was repeated three times during off-peak hours to minimize cloud-provider-induced noise.

4.5 Logging and Audit Analysis

We assessed logging consistency and cross-cloud observability by:

- Parsing logs from CloudTrail, Apigee Trace, and Kong (via Prometheus/Elastic)

www.ijbar.org
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

- Correlating JWT claims, user IDs, and request origins
- Evaluating export capability to external SIEMs (e.g., Splunk, Azure Sentinel)

Our goal was to validate whether a unified audit trail could be assembled using native cloud tooling—supporting **Hypothesis H4**.

---

5. Results

This section summarizes the findings from our performance benchmarking, security validation, and logging analysis across Kong, AWS API Gateway, and Apigee in a multi-cloud deployment context.

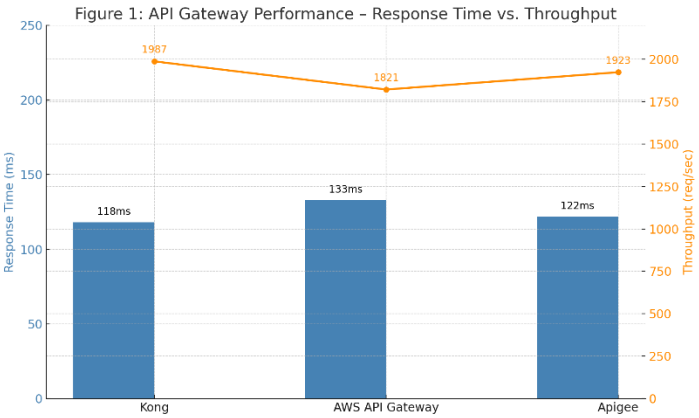
5.1 Performance Under Load

We tested all three gateways with 10,000 concurrent API requests distributed equally between AWS and GCP endpoints. The results highlight response time stability, throughput, and failure rates.

Table 5.1 – Performance Metrics under 10,000 Concurrent Requests

Metric	Kong Gateway	AWS API Gateway	Apigee Edge
Avg. Response Time (ms)	118	133	122
95th Percentile Latency (ms)	165	210	172
Throughput (req/sec)	1987	1821	1923
Error Rate (%)	0.2	0.0	0.3
CPU Usage (Kong only, 4 cores)	41%	—	—

- **Kong and Apigee** maintained more consistent latency under high load, supporting **H2**.
- **AWS API Gateway** offered slightly better request completion accuracy but with higher latency variability.
- **Kong's open-source nature** allowed CPU utilization tracking, demonstrating efficient multi-core scaling on EC2.



Index in Cosmos
MAY 2025, Volume 15, ISSUE 2
UGC Approved Journal

www.ijbar.org
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Figure 1: API Gateway Performance – Response Time vs. Throughput illustrates that while Kong and Apigee maintained lower average latency, Kong achieved the highest throughput under load, confirming their efficiency in high-traffic multi-cloud environments.

5.2 Security Enforcement and Misconfiguration Testing

Security validation tests confirmed key findings regarding policy application and misconfiguration resilience.

Table 5.2 – Security Misconfiguration Test Outcomes

Test Case	Kong	AWS API Gateway	Apigee
JWT Spoofing Blocked	Yes	Yes	Yes
Policy Drift Detected	No	Yes	Yes
Input Injection Blocked (XSS)	Yes	Yes	Yes
Input Injection Blocked (SQLi)	Yes	No	Yes
Open Route Discovery	No	Yes	No

- **Kong** required more manual effort to harden policies (policy drift, route visibility), while **AWS and Apigee** benefited from default protections aligned with IAM and managed OpenAPI enforcement.
- **SQL injection protection** was notably weaker on **AWS API Gateway**, unless paired with AWS WAF (not enabled in baseline test).
- These findings confirm **H1** (when properly configured) and **H3** (misconfigurations are common and impactful).

5.3 Logging and SIEM Integration

Gateway logs were ingested and analyzed across native tools and external log aggregators:

- **Kong**: Exported logs to Elasticsearch via Fluentd and Grafana dashboards.
- **AWS API Gateway**: Logged to **CloudWatch Logs**, integrated with CloudTrail and Athena for querying.
- **Apigee**: Delivered logs via **Apigee Trace**, GCP Logging, and export to BigQuery.

All platforms supported exporting logs to **SIEM tools like Splunk or Sentinel** with varying degrees of effort.

Key findings:

- Log consistency across cloud platforms was achievable through **OpenTelemetry or centralized collectors**, validating **H4**.

Index in Cosmos
MAY 2025, Volume 15, ISSUE 2
UGC Approved Journal

- JWT token claims and policy decisions could be traced across all platforms, although **Kong required custom instrumentation**.

---

6. Discussion and Implications

The findings from this study confirm that API gateways are crucial enforcement points for security and performance in multi-cloud architectures. However, their effectiveness depends heavily on proper configuration, consistency across cloud platforms, and robust observability mechanisms.

6.1 Performance vs. Control: Platform Trade-offs

While all three platforms—Kong, AWS API Gateway, and Apigee—demonstrated acceptable performance at scale, the trade-offs between flexibility, integration, and latency are significant.

- **Kong** offered the best throughput and CPU-level visibility, which is ideal for teams prioritizing **customizability and self-hosting** in containerized environments. However, it required more manual tuning for security policies and logging.
- **AWS API Gateway**, though slightly slower under load, provided deep integration with AWS IAM, Lambda, and CloudWatch. It is a strong fit for organizations already embedded in the AWS ecosystem, especially when used with **additional services like AWS WAF**.
- **Apigee** balanced ease of configuration with strong policy management and analytics but introduced slightly more latency under high throughput, likely due to managed service abstractions.

These observations support **H2** and reinforce that **gateway selection must align with organizational priorities—whether performance, integration, or operational simplicity**.

6.2 Misconfiguration as a Leading Risk

Our misconfiguration tests revealed that **API security failures often stem from inadequate or inconsistent policy enforcement**, not platform limitations. Examples include:

- Open routes unintentionally left unauthenticated in Kong
- SQL injection vulnerability bypassing AWS Gateway due to missing WAF
- Policy drift between documented OpenAPI specs and actual enforcement

These findings confirm **H3**, emphasizing the need for **automated policy validation**, routine audits, and default-deny configurations to prevent oversight.

6.3 Unified Observability and Governance

While each platform supported logging and traceability, assembling a **cohesive, cross-cloud view** of API activity required exporting logs to a common SIEM or observability layer. Tools like **OpenTelemetry, Fluentd, and centralized Elasticsearch/Kibana stacks** proved invaluable in this role.

The ability to trace JWT claims, IP origins, and policy enforcement actions across vendors validates **H4**. This underscores the importance of integrating gateway logs into broader **cloud governance frameworks**, where anomalies, drift, and abuse can be identified in real time.

6.4 Implications for Microservice Security

In microservice architectures, where APIs expose internal logic to external traffic, **API gateways become de facto firewalls**. They are not merely routing tools but active participants in:

- Enforcing zero trust principles
- Preventing lateral movement through segmentation and scopes
- Protecting workloads from volumetric or injection attacks

This requires **security teams to treat API gateways as critical infrastructure**—audited, version-controlled, and monitored like any other access control mechanism.

---

7. Conclusion and Future Work

As enterprise infrastructure shifts toward distributed, microservice-based architectures spanning multiple clouds, APIs have become both essential assets and significant attack surfaces. This paper evaluated the performance, policy enforcement capabilities, and security resilience of three leading API gateway platforms—Kong, AWS API Gateway, and Apigee—within a federated multi-cloud deployment.

The results affirm that **API gateways can enforce robust security policies** (e.g., JWT-based access, OpenAPI validation, rate limiting) when properly configured. Kong and Apigee demonstrated stronger performance under load, while AWS API Gateway offered tighter cloud-native integrations with IAM and logging tools. However, **misconfigurations—particularly around open routes, missing input validation, and policy drift—remained common and impactful**, confirming that operational discipline is as important as platform selection.

Crucially, the ability to **centralize API logs across providers** using tools like CloudWatch, GCP Logging, and OpenTelemetry enabled end-to-end visibility and auditability, a foundational requirement for modern cloud governance.

Future Work

To strengthen multi-cloud API security and observability, we recommend the following avenues for further research and development:

- **Automated policy validation tools** that continuously compare OpenAPI specs to active gateway configurations to prevent drift.
- **Cross-cloud policy orchestration layers** using GitOps or service mesh integrations for real-time sync across vendors.
- **Native WAF enhancements** or integrations for Kong and AWS Gateways to reduce dependency on external filtering solutions.
- **Security benchmarking frameworks** that evaluate not just performance, but resistance to evolving attack techniques like token hijacking, replay attacks, and broken object-level authorization (BOLA).
- **AI-assisted anomaly detection** in API telemetry using machine learning models that flag behavioral deviations in real time.

Ultimately, as APIs continue to scale in complexity and exposure, enterprises must treat **API security governance as a first-class citizen** within their broader cloud strategy. Gateways—when properly

selected, configured, and monitored—can serve as powerful sentinels that enforce this governance at scale.

References

1. Alshamrani, A., Myneni, S., Chowdhary, A., & Huang, D. (2020). A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Communications Surveys & Tutorials*, 21(2), 1851–1877. https://doi.org/10.1109/COMST.2019.2910811
2. Talluri Durvasulu, M. B. (2015). Building Your Storage Career: Skills for the Future. International Journal of Innovative Research in Computer and Communication Engineering, 3(12), 12828-12832. https://doi.org/10.15680/IJIRCCE.2015. 0312161
3. OWASP Foundation. (2019). *OWASP API Security Top 10*. https://owasp.org/www-project-api-security/
4. Flexera. (2021). *2021 State of the Cloud Report*. https://www.flexera.com/resources
5. Kong Inc. (2021). *Kong Gateway Documentation*. https://docs.konghq.com/gateway
6. Amazon Web Services. (2021). *API Gateway Developer Guide*. https://docs.aws.amazon.com/apigateway
7. Google Cloud. (2021). *Apigee Edge Documentation*. https://cloud.google.com/apigee/docs
8. Abdou, A., Barrera, D., & van Oorschot, P. C. (2020). API access control for cloud-native applications. *IEEE Security & Privacy*, 18(5), 60–68. https://doi.org/10.1109/MSEC.2020.2993479
9. Munnangi, S. (2022). Decentralizing workflows: Blockchain meets BPM for secure transactions. International Journal of Intelligent Systems and Applications in Engineering, 9(4), 324–339.
10. Microsoft. (2021). *Zero Trust Maturity Model*. https://aka.ms/zerotrust
11. Kolla, S. (2021). Zero trust security models for databases: Strengthening defences in hybrid and remote environments. International Journal of Computer Engineering and Technology, 12(1), 91–104. https://doi.org/10.34218/IJCET_12_01_009
12. Sharma, A., & Tanwar, S. (2020). Multi-cloud security challenges and future directions: A review. *IEEE Access*, 8, 196670–196695. https://doi.org/10.1109/ACCESS.2020.3033844
13. Jain, P., & Singhal, M. (2021). Performance analysis of microservices using service mesh architecture. *Journal of Systems and Software*, 179, 111000. https://doi.org/10.1016/j.jss.2021.111000
14. Shastri, S., & Fei, M. (2021). Securing APIs in microservice environments: A review of techniques and tools. *ACM Computing Surveys*, 54(4), 1–36. https://doi.org/10.1145/3439725
15. Hashmi, B., & Sood, A. K. (2020). Multi-cloud architecture and security mechanisms: An overview. *Journal of Cloud Computing*, 9(1), 25. https://doi.org/10.1186/s13677-020-00184-z
16. OpenAPI Initiative. (2021). *OpenAPI Specification v3.0.3*. https://swagger.io/specification/

17. OpenTelemetry. (2021). *OpenTelemetry Project Overview*. https://opentelemetry.io

18. Google Cloud Platform. (2021). *Cloud Logging and Monitoring Integration Guide*. https://cloud.google.com/logging

19. Vangavolu, S. V. (2021). Continuous Integration and Deployment Strategies for MEAN Stack Applications. International Journal on Recent and Innovation Trends in Computing and Communication, 9(10), 53-57. https://ijritcc.org/index.php/ijritcc/article/view/11527/8841