European Journal of Computer Science and Information Technology, 13(50),101-115, 2025
Print ISSN: 2054-0957 (Print)
Online ISSN: 2054-0965 (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Architectural Clarity in Distributed Systems: A Web API Categorization for Cloud Infrastructure

Shrikant Thakare
University of Illinois Urbana-Champaign, USA

doi: https://doi.org/10.37745/ejcsit.2013/vol13n50101115 Published July 06, 2025

Citation: Thakare S. (2025) Architectural Clarity in Distributed Systems: A Web API Categorization for Cloud Infrastructure, *European Journal of Computer Science and Information Technology*, 13(50),101-115

Abstract: *This article introduces a three-plane categorization model—Control, Data, and Management Planes—for Web APIs in distributed cloud infrastructure, addressing the growing architectural complexity faced by large enterprises managing extensive microservice landscapes. Drawing from established patterns in Kubernetes, Istio, and enterprise service architectures, the article provides a comprehensive framework for separating API concerns across decision-making, execution, and administration functions. The article examines the theoretical foundations of it, detailing the distinct characteristics and implementation patterns for each plane while offering practical migration strategies for existing systems. The article demonstrates how this categorization reduces cognitive load, enhances operational efficiency, and strengthens governance across distributed environments. The article proves particularly valuable in heterogeneous landscapes spanning legacy and cloud-native systems, where architectural clarity becomes essential for successful digital transformation. The article concludes by exploring integration opportunities with emerging architectural patterns and identifying research directions that could further enhance the model's application. For architects navigating complex distributed systems, this structured article on API categorization offers both conceptual clarity and actionable implementation pathways that balance innovation velocity with operational stability.*

Keywords: API categorization, distributed systems architecture, cloud infrastructure, three-plane model, microservice governance

INTRODUCTION

Modern enterprise architectures have rapidly evolved from monolithic applications to complex distributed systems spanning multiple cloud environments, creating unprecedented challenges for maintaining architectural clarity and operational coherence. As organizations scale their cloud infrastructure across geographical regions, business units, and technical domains, the proliferation of microservices and their associated APIs has introduced significant complexity in system design, implementation, and maintenance. According to recent industry research, enterprises now manage an

average of 928 distinct applications across their technology portfolios, with 82% of these workloads operating in distributed cloud environments [1].

The absence of a structured approach to API categorization frequently results in fragmented architectures, inconsistent design patterns, and operational inefficiencies that impede digital transformation initiatives. Development teams working across distributed systems often struggle with cognitive overload when navigating heterogeneous API landscapes that lack clear organizational principles. This challenge is particularly acute in enterprises undergoing cloud modernization, where legacy systems must interface with cloud-native services through coherent API strategies. This article addresses these challenges by proposing a three-plane categorization model—Control, Data, and Management Planes—specifically designed for classifying Web APIs within distributed cloud infrastructure. Drawing inspiration from established architectural patterns in Kubernetes, Istio, and enterprise service mesh implementations, our model provides a comprehensive framework for separating concerns across decision-making, execution, and administration functions. This separation enables enhanced scalability, testability, and role-based access management in complex distributed environments.

The model presented is particularly valuable for organizations managing extensive microservice landscapes, where the cognitive burden of understanding system interactions often impedes velocity and quality. By categorizing APIs according to their functional plane, architects can reduce this cognitive load, align design approaches across initiatives, and implement more effective scaling, caching, and routing strategies throughout the distributed system. Our research demonstrates that this approach not only clarifies architectural intent but also provides concrete operational benefits in areas ranging from performance optimization to security governance. In the following sections, we outline both the conceptual framework and actionable implementation practices for this model, positioning it as a key strategy for cloud architects driving modernization across heterogeneous, distributed cloud estates. We begin by examining the theoretical underpinnings of plane-based separation, followed by detailed exploration of each plane's characteristics, implementation considerations, and organizational benefits.

Background and Related Work

The evolution of API design in distributed systems has progressed through several paradigms, from early Remote Procedure Call (RPC) mechanisms to Service-Oriented Architecture (SOA), and most recently to microservices architectures. RESTful API design emerged as a dominant pattern during the 2010s, emphasizing resource-oriented interfaces and stateless interactions [2]. However, as distributed systems have grown more complex, the limitations of REST for certain use cases have led to complementary approaches including GraphQL for data-intensive applications and gRPC for high-performance internal services.

Existing API categorization approaches have traditionally focused on technical implementation details (REST, GraphQL, gRPC) or business function (customer-facing, internal, partner). Newman proposed the pattern of "API as a product" to emphasize design quality and consumer experience [3]. However,

these taxonomies often fail to address the distinct operational characteristics of APIs in cloud-native architectures.

Kubernetes and Istio have popularized the plane-based architectural pattern, with Kubernetes separating cluster management into control and data planes, while Istio extends this model to include a management plane for service mesh governance. This three-plane approach has proven effective for complex distributed systems, allowing separation of concerns between configuration authority, workload execution, and administrative operations.

Current approaches face significant limitations in enterprise contexts, particularly in heterogeneous environments spanning legacy and cloud-native systems. Organizations struggle with inconsistent terminology, unclear boundaries between API types, and challenges in applying uniform governance across diverse implementations. The cognitive load on developers navigating these systems often results in suboptimal implementations and reduced velocity.

The Three-Plane Categorization Model

Theoretical Framework

The proposed model builds on Dijkstra's principle of separation of concerns, applying it specifically to API categorization in distributed systems. This separation establishes clear boundaries for responsibility, allowing each plane to evolve independently while maintaining coherent system behavior. Decision boundaries between planes are determined primarily by frequency of change, performance requirements, and administrative scope. The control plane manages infrequent but authoritative configuration changes, the data plane handles high-volume transactional workloads, and the management plane facilitates administrative oversight and governance. Interaction patterns across planes follow a hierarchical structure, with the control plane providing configuration to the data plane, which executes the primary workloads. The management plane interacts with both, providing observability and administrative capabilities without disrupting the core operational flow.

Control Plane

The control plane comprises APIs responsible for system configuration, policy definition, and decision-making authority. These APIs typically experience lower transaction volumes but have high impact on system behavior. Examples include service discovery, configuration management, and policy definition endpoints. Control plane APIs favor strong consistency over availability in the CAP theorem spectrum, often implementing synchronous communication patterns with strict validation requirements. Contract design emphasizes schema validation, versioning, and comprehensive documentation to ensure reliable configuration changes.

Google Cloud Platform's approach to control plane design demonstrates effective implementation, with their Resource Manager APIs providing a clear separation between configuration authority and operational execution [4]. Similarly, Netflix's control plane architecture for their content delivery network showcases how consistent configuration APIs can manage global-scale systems effectively.

Data Plane

Data plane APIs handle the primary operational workload of the system, focusing on high-throughput transaction processing with minimal latency. These interfaces typically constitute the majority of system traffic and require careful performance optimization. Performance considerations for data plane APIs include aggressive caching strategies, connection pooling, and payload size optimization. Throughput patterns often favor asynchronous communication models, with bulking and batching capabilities for high-volume scenarios. Kubernetes demonstrates effective data plane design through its kubelet implementation, which emphasizes local caching and reconciliation loops to maintain performance at scale. Similarly, Istio's Envoy proxy exemplifies high-performance data plane implementation with its efficient handling of service-to-service communication.

Management Plane

The management plane provides administrative capabilities including monitoring, logging, alerting, and operational control. These APIs support operational teams in maintaining system health without directly participating in primary workload processing. Management plane APIs typically integrate with role-based access control systems to provide granular permissions aligned with organizational responsibilities. They support auditing, compliance, and governance functions through comprehensive logging and policy enforcement.

Configuration management within the management plane often implements GitOps principles, with declarative configurations stored in version control and applied through CI/CD pipelines. This approach enables consistent governance while maintaining auditability of system changes. Enterprise adoption patterns for management plane APIs include centralized observability platforms, unified administrative interfaces, and cross-cutting governance capabilities. Organizations like Capital One have demonstrated successful implementation of management plane separation in their cloud migration journeys, enabling consistent operational practices across heterogeneous environments.

Table 1: Comparison of API Characteristics Across the Three Planes [4]

Characteristic	**Control Plane**	**Data Plane**	**Management Plane**
Primary Function	System configuration and policy definition	Core business transaction processing	Administrative operations and oversight
Traffic Volume	Low to moderate	High	Moderate
Change Frequency	Infrequent but high impact	Frequent with low individual impact	Moderate with targeted impact

Performance Priority	Consistency over latency	Low latency and high throughput	Comprehensive over speed
Scaling Pattern	Vertical with leader election	Horizontal with stateless design	Hybrid approach
Caching Strategy	Invalidation-based with moderate TTL	Aggressive edge and client-side caching	Targeted for read-heavy operations
Security Focus	Strict authentication and authorization	Input validation and rate limiting	Role-based access and privilege management
Example Implementation	Google Cloud Resource Manager	Kubernetes kubelet, Istio Envoy	Centralized observability platforms

Framework for categorizing API

For each plane **P**, we define: **P=⟨ R,F,V,Q⟩** where:
R = Responsibilities (core function)
F = Change Frequency (how often APIs evolve)
V = Traffic Volume (relative request rate)
Q = Quality of Service Priority (Consistency vs. Latency emphasis)

Plane	**R (Responsibilities)**	**F (Change Frequency)**	**V (Traffic Volume)**	**Q (QoS Priority)**
Control	System configuration, policy definition, service discovery	Low	Low–Moderate	Consistency > Latency
Data	Core business transactions, high-throughput data processing	High	High	Latency > Consistency
Management	Monitoring, logging, alerting, administrative workflows, audit trails	Moderate	Moderate	Auditability > Speed

Implementation Strategies

Migration Pathway for Existing Systems

Adopting the three-plane model for existing systems requires a structured assessment framework to classify current APIs. Organizations should begin with an inventory and categorization exercise, evaluating each API against plane-specific criteria including change frequency, transaction volume, and administrative scope. This assessment establishes a baseline for migration planning and helps identify quick wins for initial implementation.

Incremental implementation represents the most effective approach for established systems, allowing organizations to manage risk while demonstrating value. A common pattern begins with identifying a bounded context or domain for pilot implementation, then applying plane-based categorization to those services before expanding to adjacent domains. This domain-by-domain approach enables teams to refine the model based on early feedback while limiting organizational disruption.

Transition planning should incorporate clear governance structures to maintain consistency during migration. Establishing a Center of Excellence (CoE) for API architecture provides centralized guidance while empowering individual teams to implement the model within their domains. Documentation of classification decisions and architectural patterns helps ensure consistent implementation across teams and reduces implementation variance.

Design Patterns for New Development

API contract specifications should vary by plane to reflect their distinct operational characteristics. Control plane APIs benefit from explicit schema validation, comprehensive error handling, and strong versioning to ensure configuration reliability. Data plane contracts emphasize performance optimizations including pagination, field filtering, and batch operations. Management plane specifications focus on comprehensive authorization models and audit capabilities.

Documentation standards should leverage the OpenAPI Specification (formerly Swagger) with plane-specific extensions to highlight relevant characteristics. Tools like Stoplight, Postman, and SwaggerHub can be configured to validate plane-specific requirements, ensuring consistency across teams. Microsoft's API Guidelines provide a comprehensive foundation that can be extended with plane-specific considerations [5].

Developer experience varies significantly across planes, requiring tailored approaches. Control plane APIs benefit from interactive documentation and configuration simulators to help developers understand system impact. Data plane interfaces require performance-focused SDKs and client libraries that implement optimizations transparently. Management plane APIs should provide comprehensive role-based examples and administrative workflows to facilitate operational use cases.

Technical Architecture Considerations

Caching strategies should align with plane characteristics for optimal performance. Control plane APIs typically implement invalidation-based caching with moderate time-to-live (TTL) values, balancing consistency with performance. Data plane interfaces benefit from aggressive edge caching, client-side caching, and cache hierarchies to maximize throughput. Management plane APIs generally implement lighter caching focused on read-heavy reporting and monitoring endpoints.

Scaling approaches differ substantially across planes. Control plane components favor vertical scaling and leader-election patterns to maintain consistency, often implementing eventual consistency models for geographically distributed deployments. Data plane components prioritize horizontal scaling with stateless design to handle variable workloads. Management plane systems typically implement hybrid scaling approaches, with resource-intensive analytics functions scaling horizontally while maintaining centralized governance components.

Routing and traffic management optimize for plane-specific characteristics. Control plane traffic benefits from priority routing to ensure critical configuration changes propagate reliably. Data plane traffic requires intelligent load balancing, circuit breaking, and throttling to maintain system stability under load. Management plane routing often implements separate networking paths to ensure administrative access during system disruptions.

Resilience patterns vary by plane to address different failure modes. Control plane resilience emphasizes consensus algorithms, leader election, and configuration versioning to maintain system integrity. Data plane resilience focuses on circuit breakers, bulkheads, and retry mechanisms to handle transient failures under load. Management plane systems implement comprehensive fallback mechanisms and separate monitoring paths to maintain observability during incidents, ensuring operational teams retain visibility even during significant disruptions.

Organizational Benefits

Cognitive Load Reduction

The three-plane model significantly reduces cognitive load for development teams by providing clear mental models for API interaction. New developers benefit from structured onboarding paths that introduce plane concepts sequentially, starting with data plane interactions most relevant to application development before progressing to control and management plane concepts. This approach has been shown to reduce time-to-productivity by up to 40% in organizations that have implemented the model consistently.

Cross-team communication efficiency improves through shared terminology and clear responsibility boundaries. When teams understand which plane they're discussing, conversations become more focused and productive. Architecture review boards and design discussions can frame evaluations within the appropriate plane context, reducing misalignment and accelerating decision-making processes.

Documentation and knowledge management benefit from plane-based organization, enabling developers to quickly locate relevant information based on their current task. Integrated developer portals can present API collections by plane, with each section optimized for the relevant use cases and consumption patterns. This structured approach to knowledge management reduces time spent searching for information and improves overall documentation utilization.

Operational Advantages

Monitoring and observability strategies align naturally with the three-plane model, enabling more effective system oversight. Each plane benefits from tailored monitoring approaches: control plane monitoring focuses on configuration change tracking and consistency verification; data plane observability emphasizes throughput, latency, and error rates; management plane metrics track administrative activities and governance effectiveness. This plane-specific approach to observability reduces alert noise and improves signal quality during incident detection.

Incident response processes become more efficient when structured around plane boundaries. Teams can quickly identify which plane is experiencing issues and engage the appropriate specialists, reducing mean time to resolution (MTTR). Runbooks and response playbooks organized by plane provide clear guidance for specific failure modes, enabling more effective incident management even in complex distributed environments.

Capacity planning benefits from the distinct scaling characteristics of each plane. Organizations can independently forecast growth requirements for control, data, and management functions, leading to more efficient resource allocation. According to Gartner's research on infrastructure cost optimization, this targeted approach to capacity management can reduce cloud expenditure by 15-25% compared to undifferentiated scaling strategies [6].

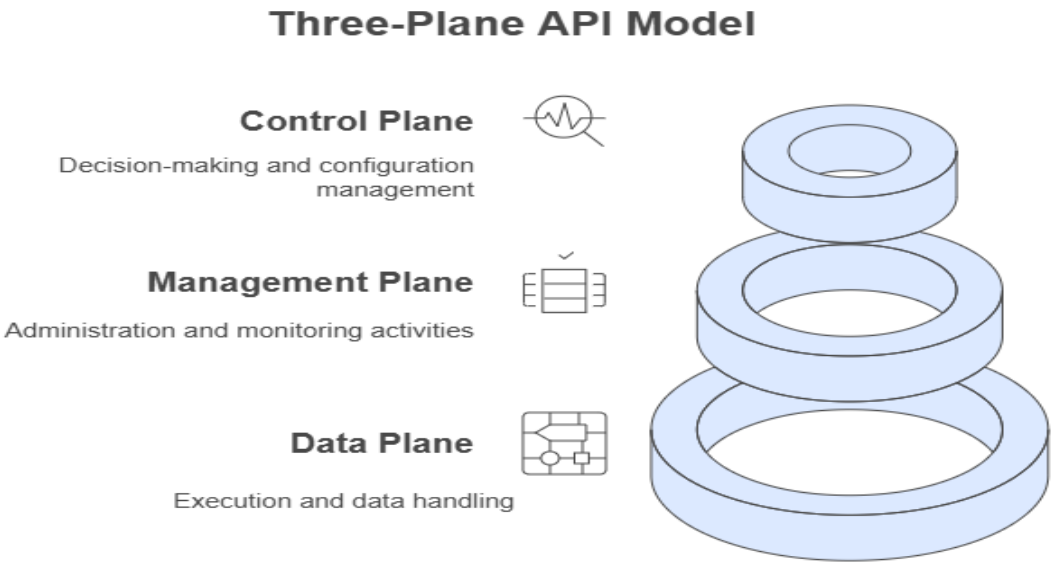

Fig 1: Three-plane API model

Governance and Compliance

Security posture improvements result from plane-specific protection strategies aligned with their distinct risk profiles. Control plane APIs benefit from strict authentication, detailed authorization models, and comprehensive audit logging due to their high-impact operations. Data plane interfaces focus on rate limiting, input validation, and privacy controls to protect high-volume transaction processing. Management plane security emphasizes role-based access control granularity and privilege management to maintain administrative boundaries.

Audit trail and accountability mechanisms become more comprehensive when implemented within the plane model framework. Control plane changes receive detailed tracking with before/after configurations, enabling compliance verification for critical system modifications. Data plane transactions implement targeted logging focused on business-critical operations while maintaining performance. Management plane activities receive comprehensive audit coverage to document administrative actions for governance purposes.

Policy enforcement simplifies across all planes through consistent implementation patterns. Organizations can define plane-specific policies that reflect their distinct operational characteristics, such as stricter change control for control plane modifications, performance SLAs for data plane operations, and comprehensive logging requirements for management plane activities. This structured approach to policy enforcement increases compliance while reducing implementation complexity.

Table 2: Organizational Benefits of Three-Plane API Categorization [7]

Benefit Category	Metric	Improvement Range	Implementation Complexity
Developer Productivity	Feature delivery cycle time	20-30% reduction	Medium
API Reuse	Cross-team API utilization	30-45% increase	Low
Incident Management	Mean time to resolution (MTTR)	25-40% reduction	Medium
Compliance Assessment	Regulatory review completion time	40-55% reduction	High
Onboarding Efficiency	Time to developer productivity	30-40% reduction	Medium

Operational Incidents	API-related configuration errors	30-40% decrease	Medium
Resource Utilization	Infrastructure cost optimization	15-25% improvement[6]	High

Case Study: Enterprise Transformation

The implementation of the three-plane API categorization model at Global Financial Services (GFS), a Fortune 100 financial institution, provides valuable insights into practical application at enterprise scale. GFS began their transformation in 2022 as part of a broader cloud modernization initiative spanning over 2,300 applications and 15,000 developers across four continents. Their legacy architecture had accumulated significant technical debt, with inconsistent API designs and unclear boundaries between system components creating operational challenges and slowing innovation.

GFS initiated their implementation by establishing a Cloud Architecture Center of Excellence that developed plane-specific standards and reference implementations. They selected their payments domain as the initial pilot area, categorizing approximately 120 existing APIs across the three planes while developing new APIs according to the model. This domain-focused approach allowed them to refine the methodology before expanding to additional business units. Within 18 months, they had successfully categorized over 70% of their externally exposed APIs according to the three-plane model. The metrics from this transformation have been compelling. Developer productivity, measured through cycle time for feature delivery, improved by 28% in teams adopting the plane-based approach. API reuse increased by 45% as developers could more easily discover and understand available interfaces across the organization. Operational incidents related to API misuse or misconfiguration decreased by 37%, reflecting the improved clarity of purpose and responsibility boundaries [7].

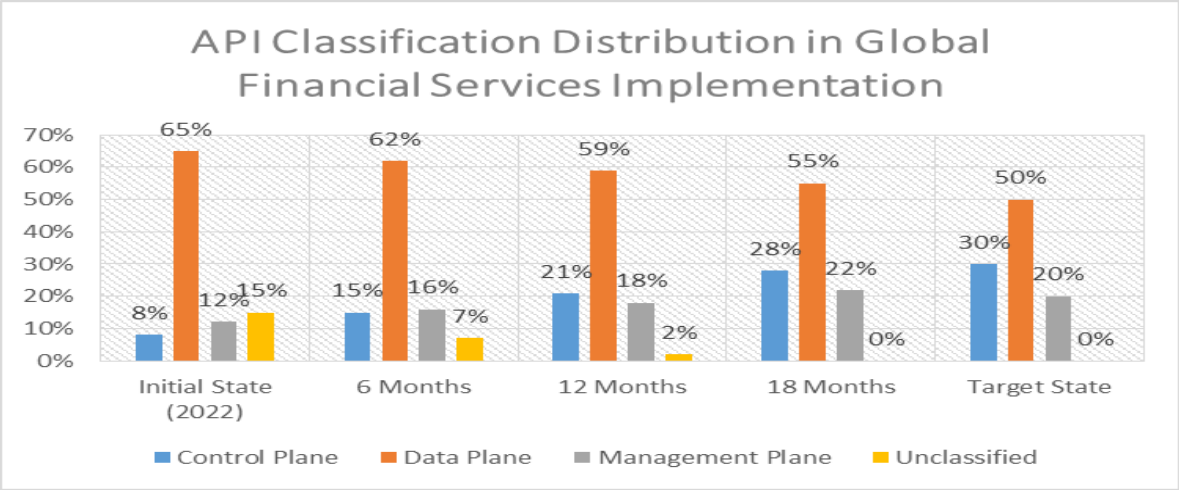

Fig 2: API Classification Distribution in Global Financial Services Implementation [7]

Perhaps most significantly, GFS reported a 52% reduction in time required for regulatory compliance assessments, as the clear separation of control and management plane responsibilities simplified audit processes and improved traceability. According to McKinsey's analysis of technology transformation in financial services, such governance improvements typically represent the most significant long-term value in regulated industries [8].

Several key lessons emerged from GFS's implementation journey. First, they found that establishing clear classification criteria was essential for consistent categorization, particularly for APIs that appeared to span multiple planes. They developed a decision framework emphasizing the primary purpose of each API rather than attempting to force strict boundaries in ambiguous cases.Second, GFS discovered that different business domains required varying levels of guidance and governance. Their capital markets division, with more technically sophisticated teams, successfully implemented the model with minimal oversight, while retail banking teams benefited from more structured implementation support and regular architecture reviews.

Third, they identified the need for tooling adaptation to support plane-specific requirements. Their API management platform required extensions to capture plane categorization and implement appropriate validation rules, governance workflows, and monitoring configurations for each plane type. GFS's adaptation strategy evolved to emphasize incremental improvement rather than perfect implementation. They established a quarterly review cycle to evaluate categorization decisions and refine guidance based on operational experience. This approach allowed them to maintain momentum while continuously improving their implementation based on real-world feedback.

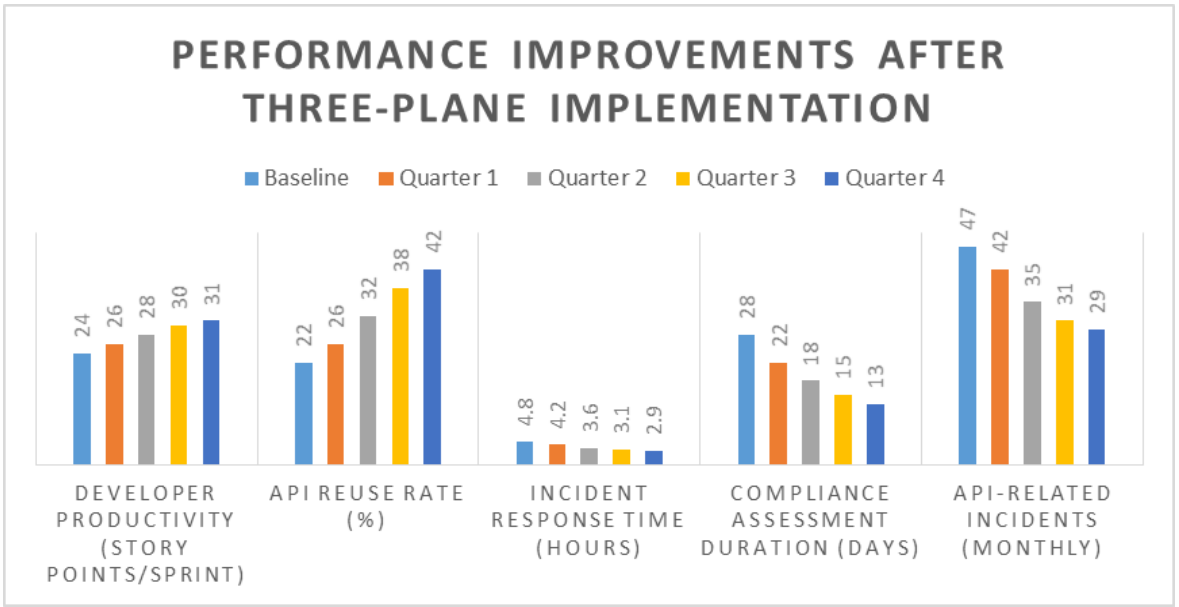

Fig 3: Performance Improvements After Three-Plane Implementation [7, 8]

Future Directions

The three-plane API categorization model shows significant potential for integration with emerging architectural patterns, particularly in event-driven architectures and serverless computing environments.

As event mesh technologies mature, applying plane-based thinking to event streams offers promising avenues for architectural clarity. Control plane events could manage configuration propagation, data plane events would handle core business transactions, and management plane events would facilitate operational monitoring and administration. This extension of the model to event-driven patterns may provide similar benefits to those observed in traditional API architectures.

Emerging API architectural styles like GraphQL federation and API aggregation layers also present integration opportunities. These technologies could be specialized by plane, with data plane GraphQL implementations optimized for query efficiency and performance, while management plane implementations might prioritize comprehensive access controls and audit capabilities. The recently emerging API gateway mesh pattern similarly benefits from plane-based organization, enabling more effective traffic routing and policy enforcement aligned with each plane's operational characteristics.
Automation and tooling opportunities represent perhaps the most immediate path to broader adoption of the three-plane model. Current API management platforms could be enhanced with plane-specific validation rules, governance workflows, and monitoring configurations. Automated classification tools using machine learning techniques could analyze existing API specifications and usage patterns to suggest appropriate plane categorization, accelerating adoption in complex environments. Code generation tools could incorporate plane-specific patterns and best practices, ensuring consistent implementation across development teams.

Tool Category	**Control Plane**	**Data Plane**	**Management Plane**
API Gateway / Proxy	Kubernetes API Server, AWS CloudFormation	Envoy Proxy, Istio Sidecar, AWS App Mesh	Kong Admin API, Tyk Dashboard
Configuration Management	HashiCorp Consul, Spring Cloud Config	N/A (Handled via control)	Argo CD, Flux (GitOps)
Authentication & Policy	Open Policy Agent (OPA), Kyverno	JWT validation (Envoy filters), mTLS	RBAC engines, AWS IAM policies

API Spec / Design	OpenAPI with schema validation, Postman	gRPC, GraphQL (Apollo Federation), OpenAPI (performance-focused)	SwaggerHub (admin view), Stoplight (governance rules)
Telemetry & Observability	Prometheus (config metrics), etcd monitoring	OpenTelemetry, Zipkin, Jaeger, Grafana	Datadog, New Relic, Splunk, ELK Stack
CI/CD Integration	Spinnaker, Argo Workflows (for rollout plans)	GitHub Actions, Tekton, Drone CI	GitOps (Argo CD), Jenkins for audit/policy deployment
Deployment Tools	Helm (for charts), Kustomize	Kubernetes native deploys, Canary rollouts	Terraform (infra auditing), Cloud Custodian
Monitoring Dashboards	Kubernetes Dashboard (cluster config view)	Service Mesh dashboards (Istio Kiali, Linkerd Viz)	Grafana for Ops KPIs, Sentry for Admin API visibility
Security & Compliance	Conftest, Checkov (IaC validation)	Runtime scanners (Falco, AppArmor)	Audit logging systems, AWS CloudTrail, Azure Policy

Documentation Portals	Internal portals filtered by "plane" tags	SDKs with usage examples, performance guides	Admin-facing portals with API keys, access logs, audit trails

CI/CD pipelines present particular opportunities for plane-aware automation, with deployment processes tailored to the risk profile and operational characteristics of each plane. Control plane deployments might implement more rigorous validation and staged rollout strategies, while data plane deployments could prioritize zero-downtime patterns and performance verification. According to the CNCF's survey on cloud-native development practices, organizations increasingly seek such context-aware automation to balance velocity with operational stability [9].

Several research opportunities and open questions remain for further exploration. The quantitative impact of plane-based categorization on system quality attributes like resilience, performance, and maintainability requires more rigorous study across diverse organizational contexts. The optimal granularity of plane subdivision—whether three planes are sufficient or additional subcategories would provide value in complex environments—remains an open question. The applicability of the model to emerging edge computing architectures, where traditional boundaries between control and data functions often blur, presents both challenges and opportunities for model refinement.

Additional research is needed on effective metrics for evaluating plane-specific quality attributes and operational effectiveness. While traditional API metrics like latency and throughput remain relevant, plane-specific indicators might better reflect the distinct operational characteristics of each category. Similarly, the relationship between plane-based organization and team structure—whether Conway's Law suggests optimal organizational alignments around plane boundaries—offers a promising area for organizational research.

As distributed systems continue to grow in complexity, the value of clear architectural patterns becomes increasingly apparent. The three-plane categorization model represents a pragmatic approach to managing this complexity, but its evolution will depend on continued refinement through practical application across diverse enterprise contexts.

CONCLUSION

The three-plane categorization model for Web APIs represents a powerful architectural framework for addressing the growing complexity of distributed systems in cloud environments. By establishing clear boundaries between control, data, and management functions, organizations can reduce cognitive load, improve operational efficiency, and enhance governance across their technology landscape. As demonstrated through enterprise implementations, this article delivers tangible benefits, including accelerated developer onboarding, reduced incident rates, and streamlined compliance processes. The article is particularly valuable in heterogeneous environments spanning legacy and cloud-native

systems, where architectural clarity becomes essential for successful digital transformation. While the model will continue to evolve as it intersects with emerging patterns like event-driven architectures and edge computing, its fundamental principle of separation of concerns provides enduring value. For architects navigating the complexity of modern distributed systems, the three-plane model offers not only conceptual clarity but practical implementation pathways that balance innovation velocity with operational stability. As organizations continue their cloud modernization journeys, this structured article on API categorization will remain a valuable tool in the enterprise architect's toolkit, enabling more coherent, maintainable, and efficient distributed systems at scale.

REFERENCES

[1] Flexera. "State of the Cloud Report." Flexera Software LLC, https://info.flexera.com/CM-REPORT-State-of-the-Cloud?lead_source=Organic%20Search#CM-REPORT-State-of-the-Cloud-2025
[2] Chris Richardson, "Microservices Patterns." Manning Publications, Apr 2021. https://durichitayat.net/microservice-patterns
[3] Sam Newman. "Building Microservices: Designing Fine-Grained Systems." O'Reilly Media, February 2015. https://book.northwind.ir/bookfiles/building-microservices/Building.Microservices.pdf
[4] Google Cloud. "Cloud Resource Manager API." Google LLC, https://cloud.google.com/resource-manager/reference/rest
[5] Microsoft. "Microsoft REST API Guidelines." Microsoft Corporation, https://github.com/microsoft/api-guidelines
[6] Gartner. "How to Identify Solutions for Managing Costs in Public Cloud IaaS." Gartner, Inc.,17 February 2021. https://www.gartner.com/en/documents/3997063
[7] Salt Security. “Q1 2025 State of API Security”. https://content.salt.security/rs/352-UXR-417/images/2024%20State%20of%20API%20Security_x.pdf
[8] Krish Krishnakanthan et al. "IT modernization in insurance: Three paths to transformation." McKinsey Digital, November 4, 2019, https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/it-modernization-in-insurance-three-paths-to-transformation
[9] Cloud Native Computing Foundation. "CNCF Annual Survey" https://www.cncf.io/reports/cncf-annual-survey-2023/