Infosys
Navigate your next

APPLICATION PROGRAMMING INTERFACES: REBUILT FOR THE NEW ERA

Infosys | Knowledge Institute

Contents

Introduction

As enterprises modernize their digital landscape, application programming interfaces (APIs) have become foundational building blocks. What began as a mechanism for system integration has evolved into a core construct for how organizations design applications, share capabilities, and scale innovation across cloud and hybrid environments.

Today, APIs shape developer productivity, security posture, and business agility. With the rise of automation and artificial intelligence (AI), their role continues to expand, reflecting a clear progression in how enterprises build, operate, and derive value from APIs over time.

API evolution: From iteration to AI-native architecture

In the first horizon (H1), organizations focused on incremental modernization through iterative service architectures. APIs and microservices were introduced as independent, reusable units layered over existing systems to enable faster integration and controlled change. Development teams adopted containers, serverless components, and DevSecOps practices to improve release velocity and developer productivity. At scale, enterprises demonstrated the ability to manage large, polyglot API estates — often spanning thousands of APIs and high transaction volumes — while maintaining stability through standardized life cycle management and governance.

As organizations progressed into the second horizon (H2), APIs evolved from simple connectors into the foundation of a cognitive ecosystem. Greater focus was on observability, event-driven interactions, and data-aware services that could adapt in real time. AI-assisted development, automated operations, and enhanced monitoring enabled teams to decompose applications into composable, reactive components. These capabilities reduced operational friction, improved resilience, and prepared enterprises for platform-based business models, where APIs coordinate interactions across applications, data, and processes.

In the third horizon (H3), organizations embedded AI directly into the API layer, creating AI-native and distributed microservices that support personalized, context-aware experiences. APIs became decision-aware interfaces, enabling semantic search, agent-driven workflows, and autonomous execution across hyperscaler and partner ecosystems. Enterprise implementations at this stage demonstrate measurable gains in response time, workforce productivity, and operational efficiency, as intelligent APIs move from supporting applications to actively shaping business outcomes in real time.

Figure 1. Adapting to market dynamics across the three horizons

	Key patterns	Characteristics
Horizon 3 **AI native APIs and distributed microservices** *Personalized and sentient*	• Agent as a service • Dynamic, runtime-bound APIs • AI gateway • MCP APIs (A2A, A2P, ACP) • Hot-swappable microservices • Distributed application runtime • Micro-gateways • AI-driven API management	• Protocol-agnostic, open specifications • Declarative, semantic APIs • Multichannel, personalized experience • Zero-trust security • AI gateways • Agentic workflows • Abstraction of complexity
Horizon 2 **Cognitive API ecosystem** *Journey from simple augmentation to being self-aware.*	• Polyglot microservices • AI-assisted development • Open source • Lightweight frameworks • Containerization • Serverless architectures • Beyond REST APIs • API gateways • Platform engineering and internal development platforms	• Opinionated, enterprise-standard driven • 12-factor app principles • Lightweight orchestration • Micro frontends • Observability and FinOps • Platform as a service • Industry standards-driven • Reactive and resilient architectures • Secure by design
Horizon 1 **Iterative-service architecture** *Continuous incremental development*	• Monolithic architectures • On-premises deployments • Process-oriented design • Strict contracts • Licensed products • Manual packaging	• Complex governance • Vendor lock-in • Agile delivery models (Scrum, Kanban, hybrid agile, and Scrumban)

Source: Infosys

Figure 2. Key trends across digital workplace services subdomains

Architecture and design	**Trend 1.** Serverless-first development enables autonomous team velocity **Trend 2.** Data mesh architecture unleashes federated data governance and organizational autonomy
Languages and ecosystem	**Trend 3.** Interest in Golang grows in lightweight microservices **Trend 4.** A unified .NET platform supports cloud-ready and AI-enabled development **Trend 5.** Java advances toward a cloud-native and AI-first platform
Frameworks and stacks	**Trend 6.** Service mesh adoption transforms microservices security and observability **Trend 7.** Polyglot cloud-native frameworks revolutionize microservices development velocity
Tools	**Trend 8.** Agent-ready API enablement becomes mainstream **Trend 9.** Observability expands to support AI-driven and agentic systems
API platforms	**Trend 10.** API-driven automation reshapes infrastructure delivery **Trend 11.** Edge clusters mature into cloud-native runtimes

Source: Infosys

Architecture and design

Enterprise development culture has evolved significantly across organizational maturity horizons. In H1, practices centered on traditional agile methodologies, strict separation of development and operations teams (silos), and time-based release cycles with limited automation. H2 introduced a transformational shift with full-stack and self-reliant teams, distributed agile practices at scale, DevOps and automation, business-driven development (BDD), low code, no code (LCNC) solutions, site reliability engineering (SRE), and financial operations (FinOps) practices, enabling organizations to respond faster to market demands.

In H3, the cultural transformation deepens with specification by example and observability at scale, enabling organizations to achieve unprecedented levels of operational efficiency, innovation velocity, and autonomous decision-making capabilities across distributed teams and systems.

Trend 1: Serverless-first development enables autonomous team velocity

Serverless computing represents a fundamental shift in how enterprises approach application development and operational responsibility. Moving beyond infrastructure management, teams can now focus exclusively on business logic while cloud providers handle provisioning, scaling, and maintenance. The architectural paradigm abstracts away the complexity of server configuration and supports organizations to iterate faster, with 64% of enterprises reporting development cycle reductions between 35% and 40%, following serverless adoption. Deployment frequency increases dramatically by 71% on average, allowing teams to deliver features and fixes to production multiple times per day without manual infrastructure provisioning or scaling concerns.

The serverless-first culture fundamentally alters team structure and decision-making autonomy. Self-reliant teams can independently design, develop, deploy, and monitor their functions without waiting for platform or infrastructure teams, accelerating feature delivery and reducing organizational bottlenecks.

This model particularly benefits event-driven architectures, stream processing for real-time analytics, and data lake construction, where teams build cloud-native applications leveraging managed serverless services as foundational building blocks.

A manufacturing enterprise that builds agricultural devices modernized its operational data pipeline using a serverless architecture, with support from Infosys. The organization implemented real-time machine telemetry processing using AWS Lambda to extract, transform, and load data from controller area network (CAN bus) interfaces into Amazon Kinesis data streams. This enabled real-time analysis, filtering, and transformation of device telemetry, with insights stored in data lakes for analytics. By eliminating infrastructure management, the serverless approach improved operational efficiency. Industry studies show that serverless data pipelines can increase deployment frequency by more than three times and reduce infrastructure costs by about 35%.

Trend 2: Data mesh architecture unleashes federated data governance and organizational autonomy

Data mesh represents a paradigm shift from centralized data engineering and governance models toward domain-driven, decentralized data ownership where business domains take responsibility for their data as products. This architectural transformation addresses the critical bottleneck of centralized data teams by empowering distributed domain teams to autonomously curate, enrich, and publish data products while maintaining compliance with enterprise governance standards. Organizations implementing data mesh achieve a 30% reduction in data latency by eliminating handoffs between central data teams and domain consumers, accelerating time to insights and enabling data-driven decision-making at the speed of business.

The cultural shift required for data mesh adoption extends beyond technology. Organizations must establish federated governance models that balance domain autonomy with centralized oversight, define clear roles and responsibilities for distributed teams, and create accountability for data quality and compliance. Enterprises recognize data mesh governance as reducing traditional bottlenecks, empowering domain teams to directly own their data pipelines rather than requesting access from overloaded central platforms. This model accelerates productivity by providing governed access to information at the right time across organizational units while ensuring data is used responsibly and complies with regulatory requirements. Implementation challenges include organizational change management, ensuring interoperability between heterogeneous data systems, and establishing data contracts that formalize the boundaries between data producers and consumers.

A global financial institution, in collaboration with Infosys, implemented a data mesh architecture to address master data management (MDM) latency challenges on legacy relational systems. The organization migrated its transactional MDM application to a high-performance distributed database, significantly reducing the latency of master data availability across the enterprise. Infosys implemented a cognitive data mapper solution leveraging machine learning (ML)-based techniques to automatically identify source-to-target attribute mapping, delivering 82% automation in value-based mapping and 60% automation in name-based mapping. This modernization enabled the bank to accelerate its enterprise data migration journey while establishing federated governance where autonomous domain teams could manage their data products independently within centralized policy frameworks, fundamentally transforming the organization's data culture from a bottleneck-driven model to an autonomous, self-service ecosystem.

Languages and ecosystem

Programming languages have evolved sharply from traditional enterprise structures to highly distributed models. Java and .NET (with C#) were the platforms of choice, anchoring large, object-oriented applications. JavaScript, with its powerful frameworks, dominated the web interface. Kotlin and Python boosted the pace of development by offering expressive, high-level alternatives for backend and data scripting, respectively, signaling the start of language fragmentation.

The necessity for speed and resilience triggered the widespread adoption of parallel and event-driven paradigms. Golang (Go) and Rust became essential for building robust, high-performance network services, with Go being the foundation for many cloud tools. The emergence of Node.js extended JavaScript's asynchronous nature to the backend, enabling quick development of real-time, event-based microservices. Enterprises fully committed to concurrency, pushing Java and .NET to enhance their core libraries with modern nonblocking and asynchronous capabilities.

The ecosystem is now defined by cloud-native, reactive, and stack-driven innovation. Languages like Kotlin and Python have been pivotal, adapting to serverless and containerized deployment with Kubernetes. Rust has cemented its role in critical infrastructure where safety is paramount. The modern, cross-platform .NET and the established Java platform have maintained relevance by streamlining their tooling for cloud deployment and AI relevance. This collective evolution has boosted the pace of innovation across the entire stack, enabling rapid development of scalable, distributed applications.

Trend 3: Interest in Golang grows in lightweight microservices

The industry is shifting toward Golang for backend services. This trend is driven by the need for green computing and FinOps (cloud cost optimization). Unlike languages that require heavy startup resources, Go compiles to machine code with a negligible memory footprint. Its goroutines (lightweight threads) allow a single server to handle thousands of concurrent processes with minimal RAM. This shift is shaping a future where compute density is maximized. Enterprises can run more microservices on smaller, cheaper cluster instances, directly reducing infrastructure bills while improving scale and reliability for high-load applications.

The migration to Go has been significantly accelerated by generative AI-driven code refactoring and transpilation. Previously, rewriting complex object-oriented logic into Go's structural, composition-based paradigms was manually intensive and error-prone. Modern large language models (LLMs) can now accurately translate verbose Java/Python boilerplate into idiomatic Go code, handling complex tasks like converting to Go's error-handling patterns. Tools like GitHub Copilot help developers bridge the skills gap, suggest correct concurrency patterns (channels and wait-groups) in real-time, effectively lowering the barrier to entry for adopting Go.

We recommend enterprises to:

1. Conduct a resource audit: Identify existing microservices with high memory to throughput ratios, such as legacy Java apps, as prime candidates for migration.

2. Pilot AI-assisted porting: Use generative AI tools to migrate a noncritical but high-traffic service to Go, allowing teams to benchmark improvements in performance and operational costs, often achieving 30% to 50% reductions in memory usage.

3. Standardize for scale: Adopt Go frameworks (like Gin or Echo) for new edge or gateway services where latency and concurrency are critical.

> A European oil and gas company partnered with Infosys to develop a next-generation DevOps and Observability platform. The company has migrated from Node.js and Python to Golang, using the Apollo library to expose GraphQL endpoints, for its new set of microservices optimized for efficient memory management capabilities.

Trend 4: A unified .NET platform supports cloud-ready and AI-enabled development

The fragmentation of the past (splitting .NET Framework, Core and Xamarin) is over. The industry is rapidly standardizing on a single .NET runtime (currently .NET 8/9/10) that delivers uniform behavior across cloud, desktop, and mobile. This unification provides a streamlined unified .NET SDK experience, where one set of tools builds everything from web frontends to high-performance background workers. A key driver here is .NET Aspire, an opinionated, cloud-ready stack that simplifies orchestrating distributed applications, handling observability, and wiring up dependencies like Redis or PostgreSQL automatically.

The future is portable high performance: Developers can now ship self-contained, single-file microservices using Native ahead-of-time (AOT) compilation, which start instantly and deploy easily to Linux containers, leveraging HTTP/3 and faster base case library algorithms for maximum throughput.

The unified .NET trend is heavily reinforced by new libraries that allow developers to build modern AI workloads without leaving the ecosystem or relying on external Python bridges. Semantic Kernel is an open-source SDK that enables C# developers to integrate LLMs directly into their applications. It orchestrates AI plugins, which are simply native C# functions, allowing the .NET runtime to serve as the brain for complex AI agents. The .NET base class library has also been updated with support for tensors. This provides low-level, hardware-accelerated support for mathematical operations used in AI. It powers libraries like ML.NET and TorchSharp, enabling the unified runtime to execute deep learning models and local inference efficiently across any supported platform.

We recommend enterprises to:

1. Adopt .NET Aspire: Make Aspire the default starting point for all new cloud-native projects to bake in observability and service discovery from day one.

2. Leverage Native AOT: For lambda functions or sidecar containers, switch to Native AOT compilation to drastically reduce image size and cold start times.

3. Consolidate tooling: Retire legacy specific build pipelines and standardize on the unified .NET command-line interface (CLI) to build, test, and publish containers across all environments.

An American multimodal logistics company has partnered with Infosys to build its digital freight shipping management system, designed to streamline the end-to-end life cycle of its transportation operations. The solution is built on generative AI capabilities in Google Cloud and the unified .NET stack, providing a consistent, high-performance foundation across all services, including real-time quotes, order management, carrier allocation, route optimization, and shipment tracking.

Trend 5: Java advances toward a cloud-native and AI-first platform

Java is consolidating into a cohesive application platform centered on Java 25/26, Spring Boot 4.0, and Jakarta EE 11. This shift reduces long-standing complexity while enabling cloud-native execution, AI-native integration, and high-performance workloads, positioning Java for both enterprise modernization and next-generation application development.

At language and runtime levels, virtual threads (permanent since Java 21) support massive I/O concurrency using familiar blocking code, reducing reliance on reactive models. Combined with structured concurrency, they simplify safe parallel execution and are now viable for mainstream microservices and APIs through Spring Boot 3.2+.

Java's performance model is evolving through Project Valhalla, introducing value types to reduce object overhead, improve memory locality, and lower garbage-collection pressure. Planned generic enhancements aim to eliminate boxing for primitives. While not yet production-ready, Valhalla represents a foundational change expected in Java 26-27.

The runtime layer is being modernized via AOT compilation with GraalVM Native Image and Project Leyden, addressing startup and memory constraints. This expands Java's suitability for serverless, event-driven, and container-dense workloads, supported by frameworks such as Quarkus, Micronaut, and Spring's native tooling.

Java is also moving toward AI-native application development. Spring AI and LangChain4j provide abstractions for LLMs, embeddings, vector databases, and RAG patterns, while the Vector API provides SIMD-optimized operations for AI-adjacent processing and inference.

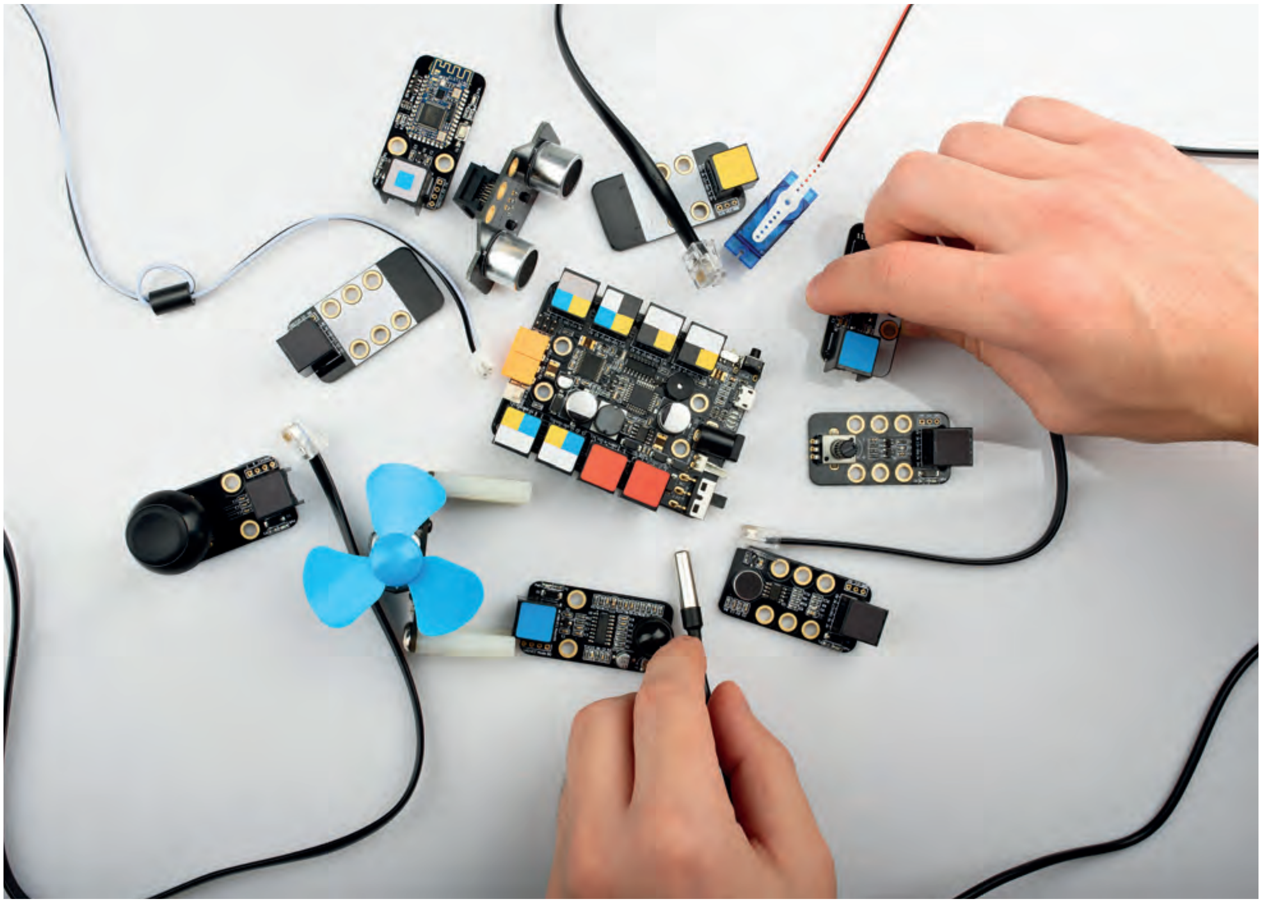
The ecosystem is aligning with platform engineering practices. Spring Boot 4.0 and Spring Framework 7.0 enhance API versioning, resilience, and null-safety, while Jakarta EE 11 provides a stable enterprise baseline. Together, these support internal developer platforms (IDPs), golden-path architectures, and Kubernetes- and Spring Cloud-based deployments.

Enterprise modernization remains a key driver. Migrations from Java 8 and Java 11 unlock modern language features such as records, pattern matching, sealed classes, and text blocks, with OpenRewrite increasingly used to automate upgrades at scale.

We recommend enterprises to:

1. Adopt Java 21+ as a baseline and use automated refactoring for large-scale upgrades.
2. Trial virtual threads for I/O-bound microservices and high-concurrency APIs.
3. Evaluate GraalVM Native Image for serverless and container-optimized workloads.
4. Assess Project Valhalla for future data-intensive and analytical systems.
5. Explore Spring AI and LangChain4j for governing generative AI integration.
6. Standardize platform engineering patterns using Kubernetes, Spring Cloud, and IDPs.

Frameworks and stacks



The evolution of frameworks and technology stacks has fundamentally reshaped enterprise application development. In H1, organizations relied on traditional model-view-controller architectures and monolithic web services with limited integration patterns, using legacy frameworks like Zend and ColdFusion, basic XML/JSON serialization, and traditional SOAP-based security models. H2 introduced full-stack frameworks optimized for memory efficiency and microservices patterns, embracing technologies like GraphQL, Spring Cloud, Apache Camel, and containerized deployments with enhanced monitoring through OpenCensus and OpenTracing. H3 represents a paradigm shift toward reactive, cloud-native architectures powered by frameworks such as Quarkus and Micronaut, advanced messaging systems like Apache Pulsar and NATS, high-performance data serialization with Protobuf and Avro, and sophisticated service mesh technologies including Istio and Cilium. These modern stacks deliver unprecedented levels of performance, scalability, and resilience for next-generation distributed systems.

Trend 6: Service mesh adoption transforms microservices security and observability

Service mesh architectures have emerged as a critical infrastructure layer for managing complex microservices ecosystems at enterprise scale. Istio, as the most prominent cloud-native computing foundation graduated service mesh implementation, addresses fundamental challenges including zero-trust security, advanced traffic management, and comprehensive observability without requiring application code changes. The architecture separates infrastructure concerns from business logic through sidecar proxies deployed alongside each microservice, enabling uniform policy enforcement across heterogeneous technology stacks. Organizations implementing service mesh report significant improvements in operational resilience, with automated traffic routing, health checks at the pod level, and seamless blue/green deployments that minimize downtime during frequent production releases.

Security represents a primary driver for service mesh adoption. Istio provides workload identity-based zero-trust security through mutual transport layer security (mTLS) encryption between all service-to-service communications, eliminating the need for developers to implement encryption logic within applications. Multilayered authorization policies validate JWT tokens at ingress gateways and individual pod sidecars, creating defense-in-depth protection against unauthorized access. This standardized security model proves particularly valuable for financial services and regulated industries where compliance requirements mandate encrypted communication channels and granular access controls across distributed systems.

Traffic management capabilities in a service mesh enable sophisticated deployment patterns including canary releases, A/B testing, and load-based routing with autoscaling. Virtual services and destination rules specify routing logic declaratively, allowing operations teams to dynamically adjust traffic flow between service versions without modifying application code or redeploying containers. Combined with comprehensive observability through Kiali dashboards for real-time traffic visualization, Jaeger for distributed tracing, and integration with monitoring platforms like AppDynamics and ELK stack, service mesh delivers end-to-end visibility into microservices behavior that traditional monitoring approaches cannot achieve.

Infosys partnered with a large financial institution to implement Istio service mesh on OpenShift, architecting a multilayered security solution for a mission-critical application processing billions of dollars in transactions. The team deployed Azure AD authentication at the perimeter, JWT token validation across all service boundaries, and ingress gateway protection blocking unauthorized hosts. Istio's authorization policies enforced fine-grained access controls uniformly across all microservices pods through sidecar injection. The implementation combined traffic management with Kubernetes autoscaling, enabling the system to handle increased transaction volumes without delays or failures. Observability was achieved through Kiali dashboards for real-time traffic visualization, Jaeger integration for granular API monitoring, ELK stack for centralized logging, and AppDynamics agents for custom metrics and alerting. The solution eliminated decades-old silos, achieved a 40% increase in operational efficiency and cost savings, generated $300 million in revenue opportunities, and maintained zero security incidents throughout production operations. The robust service mesh architecture enabled seamless blue/green deployments despite frequent software upgrades, driven by evolving business requirements.

Trend 7: Polyglot cloud-native frameworks revolutionize microservices development velocity

Modern polyglot Java frameworks including Quarkus, Micronaut, and Helidon have emerged as transformative technologies for cloud-native and serverless application development, addressing critical limitations of traditional enterprise Java stacks. Quarkus achieves millisecond startup times through compile-time dependency injection, classpath trimming, and AOT compilation via GraalVM, enabling seamless deployment to serverless platforms like AWS Lambda and Azure Functions without cold-start performance penalties typical of traditional JVM applications. The framework's reactive programming model supports nonblocking I/O operations and concurrent session handling at massive scale, critical requirements for modern microservices handling fluctuating traffic patterns, and resource constraints of containerized environments.

Micronaut takes a complementary approach, emphasizing compile-time annotation processing to eliminate runtime reflection overhead, resulting in applications with minimal memory footprint and rapid startup regardless of codebase size. Both frameworks support multiple execution models — imperative, reactive, and virtual threads — enabling developers to select the optimal programming paradigm for specific use cases without framework constraints. These modern frameworks standardize on CloudEvents for event schema definition, enabling vendor-neutral event communication across heterogeneous systems, while integrating seamlessly with NATS and Apache Pulsar for high-performance message delivery. The shift from language and framework selection toward holistic stack optimization reflects market maturation, where developers build composable systems leveraging native end-to-end synergies among frameworks, messaging systems, serialization formats, and deployment platforms.

The adoption of polyglot frameworks accelerates development velocity by eliminating boilerplate infrastructure code. Teams deploying microservices on Kubernetes using Quarkus or Micronaut report 35% increased deployment frequency and 20% faster lead time for changes, directly translating to competitive advantage in rapidly evolving markets. These frameworks' native support for observability, security, and resilience patterns, including distributed tracing, health checks, and circuit breakers, shifts responsibility from application developers to the framework, allowing teams to focus exclusively on business logic rather than operational concerns.

A leading financial services institution partnered with Infosys to modernize its payment services platform using a modern, cloud-native technology stack. The organization developed the GRAND stack framework (GraphQL, Reactive, and modern Java frameworks combined with next-generation databases) to replace legacy monolithic payment processing systems.

The GRAND stack architecture delivered millisecond-level authorization response times, 99.9999% availability for critical payment flows, and enabled rapid feature deployment for new payment modalities, including digital wallets and real-time payments. The framework eliminated dependencies on individual programming language choices, enabling cross-functional teams to contribute while maintaining consistent operational characteristics and performance guarantees.

Tools

Enterprises are rapidly moving toward agent-ready API ecosystems, where traditional REST and GraphQL services are refactored to be directly consumable by AI agents. Toolchains such as Google's Agent Development Kit repurpose existing OpenAPI specifications into agent-usable tools, while Apollo's GraphQL model context protocol (MCP) server exposes schema operations in the same manner. This shift signals a broader trend in which APIs are no longer just machine-readable interfaces, but semantically rich, self-describing capabilities optimized for autonomous agent workflows.

At the platform layer, AI gateways and semantic toolchains are becoming the new control plane for LLM-driven systems. Enterprises now route every model invocation through specialized gateways that provide governance, security, cost optimization, and semantic enrichment via vector caches, prompt guards, and intelligent routing. These capabilities are reinforced by advances in AI observability, where traditional application performance monitoring (APM) expands to include prompt tracing, token-level metrics, safety drift detection, and end-to-end auditability. Cloud providers and open ecosystems, such as Amazon CloudWatch's generative AI observability enhancements and OpenTelemetry's emerging semantic conventions, are establishing the standards for monitoring agentic and LLM-enabled applications.

Aligned with these shifts, modern API and microservice tooling are being reshaped by lightweight execution environments, LCNC automation, and AI-augmented testing. WASM runtimes enable ultra-fast, portable microservices with near-zero cold starts, exemplified by platforms like Fermyon Spin. Integration suites such as Boomi, MuleSoft, and Informatica now blend drag-and-drop orchestration with embedded AI gateways and service-mesh policies, automating secure data flows and PII handling. Meanwhile, API quality engineering is being transformed through AI-driven test generation and automated failure triage, with platforms like Postman introducing agents such as PostBot to streamline debugging, documentation, and coverage expansion.

Trend 8: Agent-ready API enablement becomes mainstream

Agent-ready API enablement has recently evolved into a more structured and scalable approach for connecting AI agents with backend services. A key enabler is OpenAPI Toolset (part of Google Agent Development Kit/ADK), which can automatically transform a standard OpenAPI specification into callable tools (e.g., RestApiTool) that AI agents can invoke directly, removing the need for manual wrapper coding. On the GraphQL side, Apollo MCP Server has matured into a robust production-grade solution for exposing GraphQL operations as agent-consumable tools under the MCP standard. GraphQL queries and mutations defined in the schema or persisted-query manifests are automatically available to any MCP-capable client, giving agents structured access to data and business logic without bespoke integration. Beyond these, newer frameworks are emerging that further streamline agent-to-API workflows. For example, tools like FastAPI-MCP enable Python-based REST services to be exposed as MCP servers with minimal configuration, making them instantly usable by agents. There is also growing emphasis on dynamic tool discovery and selection, as demonstrated by frameworks such as ScaleMCP, which allows agents to retrieve and register tools at runtime, reducing overhead and avoiding redundant tool-repository maintenance.

> A leading multicountry quick-service restaurant operator modernized its finance operations through an agentic AI-enabled, cloud-based accounts payable platform that autonomously processes invoices end to end, with multilingual support, complex validations, and minimal human intervention. Infosys supported the transformation, which improved accuracy and efficiency while enabling predictive insights, adaptive processing, and continuous learning powered by Azure GPT-4o.

The growing adoption of agentic AI is driving API platforms to evolve beyond traditional integration patterns and offer interfaces that are inherently agent-aware. GraphQL and orchestration ecosystems now provide native mechanisms for exposing structured capabilities, streamable interactions, unified runtimes, and built-in telemetry without requiring additional wrapper code. Emerging research adds dynamic tool-discovery and synchronization, enabling agents to efficiently identify and use available capabilities on demand.

Trend 9: Observability expands to support AI-driven and agentic systems

The rise of AI-powered applications and agentic workflows has driven observability platforms to evolve far beyond traditional APM. Modern AI observability tooling now captures not only infrastructure metrics but also prompt-level traces, token usage, model invocation latency, error rates, and downstream embedding drift or safety signals. For instance, Amazon CloudWatch now offers a dedicated generative AI observability capability that delivers out-of-the-box dashboards tracking latency, token consumption, errors and model usage — and crucially, supports end-to-end prompt tracing across models, knowledge bases, tools, and agent workloads. Beyond CloudWatch, observability is increasingly standardized around OpenTelemetry (OTel), now extended with generative AI semantic conventions and instrumentation libraries that automatically capture telemetry for LLM-based applications — including prompts, completions, tool calls, token counts, and cost metrics. Vendors such as IBM Instana have released generative AI observability sensors that leverage OTel to instrument full AI stacks (models, agents, runtimes, infrastructure), offering integrated tracing, performance monitoring, alerting, and cost/resource analytics. These developments facilitate organizations to deploy generative AI and agentic systems in production with high confidence in reliability, observability, and operational governance.

To address cost predictability of agentic AI implementations, Infosys Topaz Agent ROI cost calculator has been designed to provide flexible and accurate agent runtime and build cost estimation.

API platforms as a service

API platforms have changed a lot over the last decade. The initial focus was on enabling modular software development and faster release cycles. This has now evolved into creating a foundational layer for enterprise-scale digital ecosystems. Traditional API gateways and service-oriented architecture have been replaced by cloud-native platforms, event-driven models, and service meshes that offer granular control, resilience, and observability. Today, platforms handle self-optimization, threat prediction, adaptive scaling, and automated policy enforcement. Along with the acceptance and adoption of hybrid and multicloud, platforms are evolving to support portability, security, and real-time data interchange across diverse environments.

AI is no longer an afterthought. Platforms have become AI-native to enable services as intelligent autonomous units, capable of taking reactive and proactive measures to meet scalability, performance, and security requirements.

Trend 10: API-driven automation reshapes infrastructure delivery

Organizations are increasingly building internal PaaS-style platforms that expose APIs for development, continuous integration and continuous delivery/deployment (CI/CD), and infrastructure orchestration. These internal marketplaces provide a simplified, often opinionated abstraction over cloud or third-party SaaS services, while giving the organization control and insights into the use of these services, which can be critical when dealing with security vulnerabilities and compliance aspects.

Declarative infrastructure, i.e., infrastructure a code (IaC), combined with version-controlled workflows (GitOps), has become the de facto standard for automated provisioning and deployment. This approach manages infrastructure through high-level configuration files that describe the desired state of resources, rather than step-by-step scripts. Common examples are Terraform scripts and Kubernetes manifests. Policy as code (PaC) is increasingly used to enforce compliance, security, and operational policies automatically, with tools such as Open Policy Agent and Kyverno.

AI and ML are being embedded into API management and infrastructure tooling. For example, automating API design, predictive autoscaling, anomaly detection, and even self-healing infrastructure. API gateways, which were once simple proxies, now deeply integrate with service meshes and manage capabilities like authentication and authorization, routing, rate limiting and protocol translation. Service meshes handle interservice (east-west) traffic with secure communication (mTLS), traffic management, and resilience.

Zero-trust security principles and fine-grained governance at the API and infrastructure layers are being widely adopted. Authorization policies are enforced automatically through policy-as-code and service mesh controls.

> A leading conversational AI company serving Fortune 500 clients implemented platform engineering capabilities for customer support and e-commerce chatbots, with support from Infosys. The solution included a self-service developer portal, automated infrastructure provisioning using IaC, and integrated observability tools for real-time monitoring. Standardized CI/CD pipelines and Kubernetes-based container orchestration reduced deployment time by 60%. This approach enabled faster feature delivery and improved reliability and scalability across multiple workloads.

Trend 11: Edge clusters mature into cloud-native runtimes

Edge clusters are maturing into lightweight, cloud-native runtimes. Organizations run small Kubernetes flavors (k3s and the like) or purpose-built orchestrators onsite to host containerized microservices close to users and devices. This improves millisecond-level responsiveness while keeping central control planes. Functions at the edge-driven and event-driven microservices let teams push only the minimal processing to edge clusters, lowering cold-start costs with prewarmed runtimes and speeding per request latencies for APIs.

Frameworks and platforms catering to this space are provided by both major cloud providers (AWS IoT Greengrass, Azure IoT Edge) and the open-source space (Kubeless, Apache OpenWhisk, FaaS, Knative).

Telecom operators are deploying multiaccess edge computing nodes to support ultra-low-latency APIs, allowing service providers to act as regional cloud and edge operators. As AI moves closer to users and devices, micro clusters equipped with graphics processing units and application-specific integrated circuits support use cases such as video analytics and predictive maintenance, running inference at the edge to maintain predictable response times when a high-bandwidth link to a cloud data center is not feasible.

Advisory council

Mohammed Rafee Tarafdar

Dinesh Rao

SP Singh

Manas Sarkar

Vishwanath Taware

Authors

Shreshta Shyamsundar

Sudheesh Nair

Joseph Alex

Arun George

Rashmi Bhojwani

Contributors

Charudatta Joshi

Lakshminarayana Indraganti

Vikas Thakur

Dinesh Nagabushanam

Naveen Satrasala

Ramanath Shanbhag

Sachin Waman Junghare

Badri Prasad S

Nilesh Ashok Ghone

Vijendra Rajput

Sriram P

Producers

Ramesh N
Infosys Knowledge Institute

Pragya Rai
Infosys Knowledge Institute

About Infosys Knowledge Institute

The Infosys Knowledge Institute helps industry leaders develop a deeper understanding of business and technology trends through compelling thought leadership. Our researchers and subject matter experts provide a fact base that aids decision-making on critical business and technology issues.

To view our research, visit Infosys Knowledge Institute at infosys.com/IKI or email us at iki@infosys.com.

For more information, contact askus@infosys.com