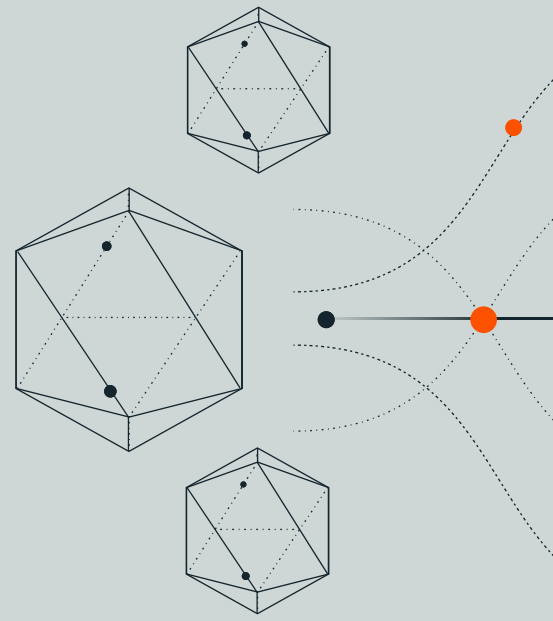
APOLLO

SOLUTION BRIEF

Breaking monolithic barriers: Apollo GraphOS and the journey to microservices driven innovation



KEY BENEFITS

TRANSITION AT YOUR OWN PACE

A unified API composition layer that connects APIs across any cloud, orchestrator, platform, or protocol and easily integrates and orchestrates new and legacy services across clients.

MAXIMIZE DEVELOPER EFFICIENCY

Speed up feature delivery and safely migrate away from the monolith architecture with decoupled frontend and backend development.

REDUCE TOTAL OPERATIONAL COST AND OVERHEAD

Maximize efficiency and lower operational expenses by minimizing extensive backend coding, and optimizing resource allocation through a modern, modular architecture.

Introduction

Application modernization is imperative for organizations looking to enhance agility, improve scalability, and foster innovation. At the heart of the modernization initiative lies the essential process of refactoring monolithic applications into microservices. Unlike monolithic architecture where components are tightly coupled, microservices architecture breaks an application into smaller, loosely coupled services, allowing scalability, resilience, ease of deployment, and technology diversity. This transition from monolithic architectures to microservices is not just a trend but has become a necessity. In fact a survey finds **92%** of enterprises are working on or planning an app modernization project. And yet **70%** of application modernization projects fail.

Why?

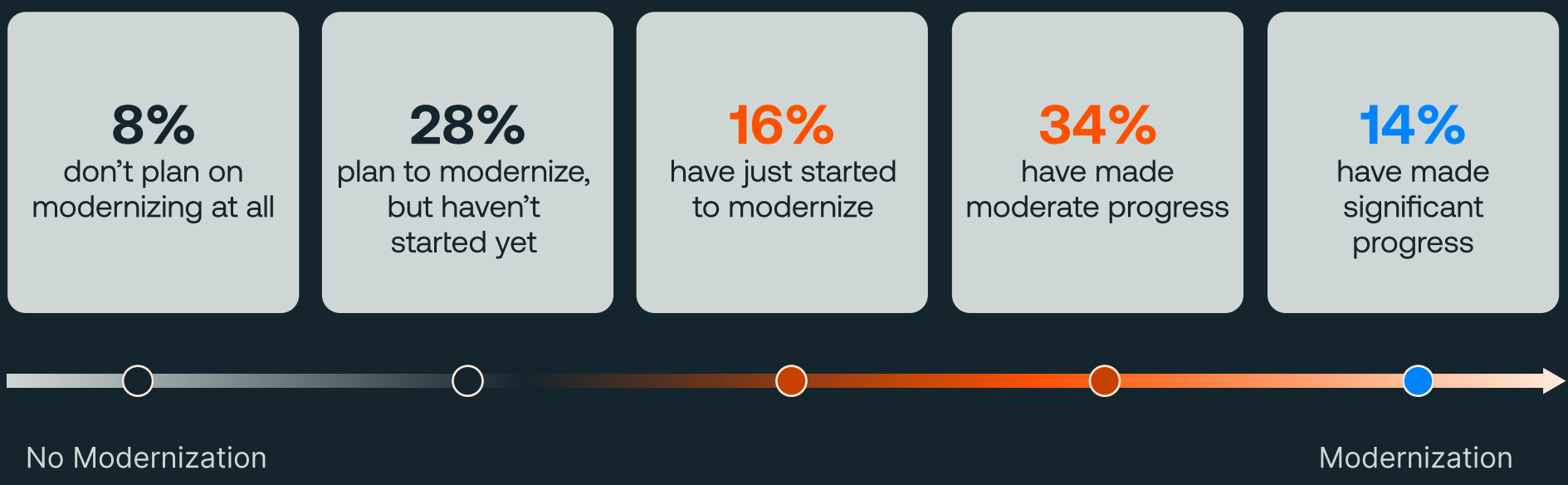


Figure: The State of Modernization[1]

One major challenge contributing to these failures is the complexity associated with managing a growing number of Application Programming Interfaces (APIs). A recent Postman survey indicated that **49%** of respondents said most of their organization's development effort was spent working with APIs. In a microservices architecture, each service communicates through these API contracts, which can number in the hundreds or even thousands for a single application. A single client request may require hundreds of API calls to different services, making the frontend teams face the daunting task of aggregating, orchestrating, and composing APIs.

The highly distributed and dynamic nature of the microservices architecture has necessitated a shift in how teams operate. Developers can now build services in various programming languages, protocols and using different data storage technologies. While this flexibility allows developers to tailor solutions to meet the needs of customers and organizations, each option brings its own set of APIs and interaction protocols, complicating the integration and consistent API management across different services. Cross-cutting concerns, such as security, governance, and monitoring need to be applied to each microservice, and the functionality must be replicated for each language used in the application. This often leads to redundant implementations and inconsistencies between services, forcing developers to write extensive code changes to ensure seamless connectivity between old and new services. This added effort makes it increasingly difficult to iterate and deliver new features rapidly.

Moreover, modernization is not an overnight transformation. For successful modernization, organizations need a resilient API strategy that meets their modern architectural requirements while supporting their legacy environments.

Exploring API strategies

While enterprise API programs have matured in terms of stability, security, and discoverability through tools like API gateways, the true value emerges when these APIs are effectively utilized across various frontend applications. To deliver on this half of the equation, developers are on the hook for the hard work to aggregate, orchestrate, and compose APIs into something that is useful and discoverable for client teams – the last mile of APIs.

[1] "WHY APP MODERNIZATION PROJECTS FAIL." VFUNCTION, 2022, https://vfunction.com/resources/report-wakefield-why-app-modernization-projects-fail/

Traditional approaches like backends-for-frontends (BFFs) or Experience APIs aim to abstract backend complexity for frontend teams, but they often lead to duplicated effort, inconsistencies across frontends, as well as technical debt, ultimately hindering innovation.

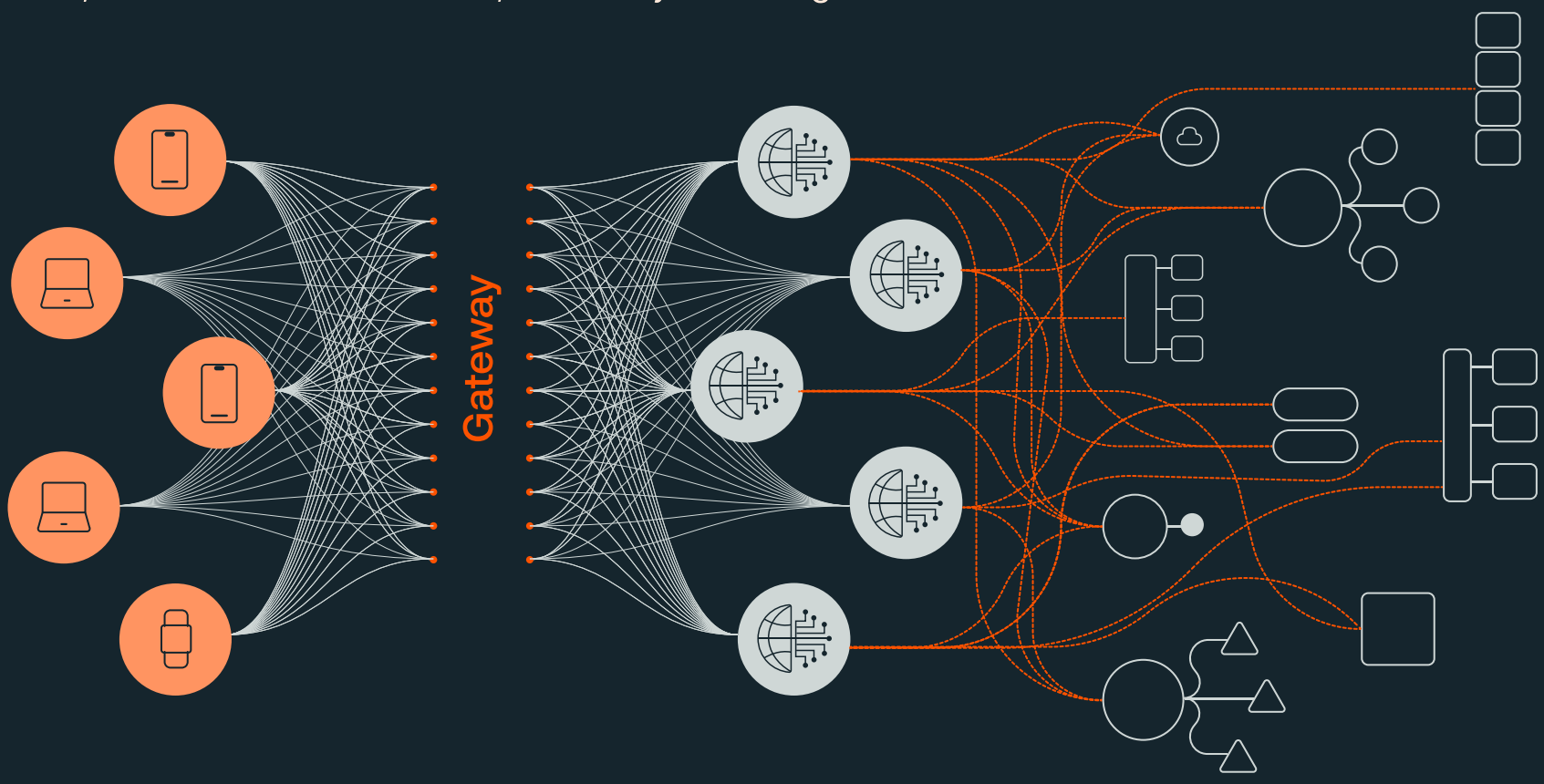


Figure: Experience APIs or backends-for-frontends

Apollo GraphOS: The API platform for the modern stack

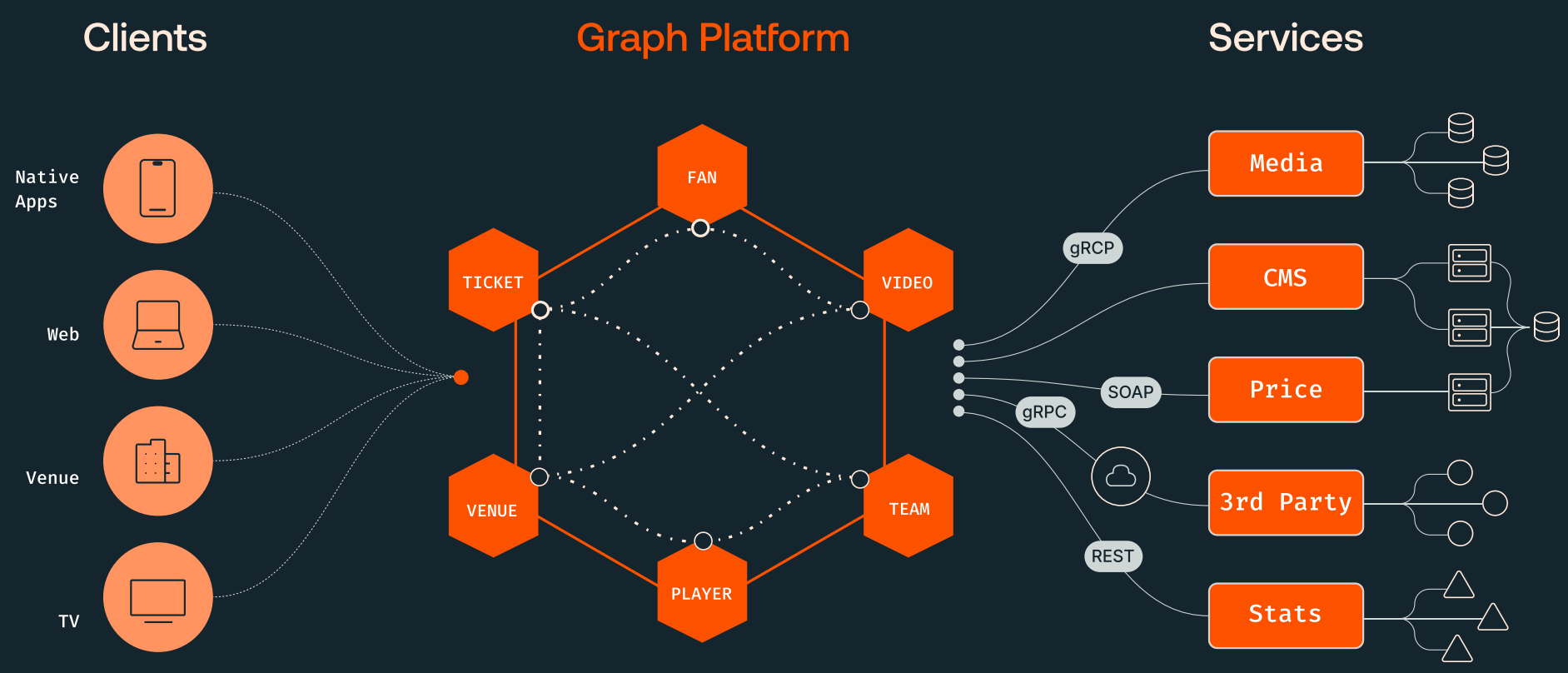


Figure: GraphQL platform – an API composition layer

Future-proof your API strategy while magnifying your existing API investments

Platform engineering is an emerging trend intended to modernize enterprise software delivery, and provides immense benefits including self-service, automation, standardization, centralized security, and composability. This enables software teams to ship features faster and support any number of modernization initiatives with less work. Apollo GraphOS plays a critical role in extending these benefits at the API layer by introducing a composable abstraction layer. Rather than delivering APIs as a barrage of new endpoints to manage, a federated GraphQL architecture (Apollo Federation) powered by Apollo GraphOS provides a unified API experience.

The GraphOS platform can meet you wherever you are on your modernization journey as it empowers teams to use GraphQL to build an API composition layer that sits on top of REST, SOAP, or any other legacy services, helping magnify an organization's existing API investments. This composable abstraction layer helps map data in a consistent and flexible way, regardless of who manages it and what kind of infrastructure or API protocol is working behind the scenes. This technology-agnostic approach simplifies integration, allows easier adoption of new technologies, and reduces dependency risks.

Maximize developer efficiency

The modular, federated GraphQL architecture also helps to decouple frontend and backend development. It enables backend teams to choose the best technology to build a service by providing the ability to connect their APIs across any cloud, orchestrator, platform, or protocol and make them available via a single endpoint. The decoupled architecture empowers backend teams to safely and predictably migrate away from the monolith, without impacting clients – ultimately reducing tech debt. Frontend teams can work independently, designing queries that suit their UI requirements without direct dependencies on backend changes. Eliminating friction between frontend and backend teams helps deliver features faster and fosters a culture of experimentation, allowing frontend developers to prioritize user needs over API versioning and data stitching concerns.

Reduce operational costs and overhead

Apollo GraphOS enables organizations to efficiently decouple the complexity of legacy systems by shifting the logic of aggregating data and managing BFF layers into a declarative approach with GraphQL. This means that the data aggregation, orchestration, and composition logic, typically hardcoded on the client side or in BFFs, can now be seamlessly managed through GraphQL's powerful and flexible architecture. This approach reduces the need for additional coding, simplifies the overall architecture, and enhances maintainability by centralizing the logic in a more manageable and scalable manner. Apollo GraphOS decouples the connectivity concerns from the application developer's workflow so they can focus on building new features, optimizing their development efforts, and cost.

Case study: PayPal

PayPal, a pioneer in digital payments, has significantly enhanced its service delivery by adopting Apollo's Graph Platform, transforming the digital payment landscape for over 300 million users worldwide. Faced with challenges in developer productivity and end-user performance using traditional REST APIs, PayPal's engineering team adopted GraphQL, achieving remarkable improvements. This strategic move reduced unnecessary data fetching complexities, enabling developers to focus more on user interface creation and less on backend data management. A graph was also incredibly straightforward to implement. PayPal built the graph at the edge of their stack that leveraged existing core REST APIs. There was nothing to rip-and-replace, and they could migrate away from legacy infrastructure over time.

As PayPal expanded its use of the graph — not only supporting more products, but also integrating with internal systems and processes such as authentication and authorization — the team quickly realized that they needed a team and proper tooling to manage the graph.

They began using the Apollo GraphOS to streamline data integration and management across more than 50 products, simplifying operations without compromising performance. A central team manages PayPal's graph with Apollo, ensuring it's protected, monitored and managed.

This modernization initiative supported by Apollo GraphOS has positioned PayPal at the forefront of financial technology, delivering faster, more reliable payment solutions that enhance user satisfaction and drive business growth.

Case study: Expedia

Expedia Group, a leading travel platform managing over 200 booking sites and 25 brands, faced challenges due its complex technology stack. The company struggled with the task of delivering a seamless experience for travelers using multiple touchpoints during their planning, shopping, and traveling journeys. Seeking to improve customer experiences and streamline their development process, Expedia embarked on a journey to adopt a unified graph approach with GraphQL and Apollo. This transformation was driven by the realization that their existing architecture, comprising multiple traditional REST services, led to bottlenecks, duplication of effort, and slowed down feature delivery. By transitioning to a federated GraphQL architecture powered by Apollo GraphOS, Expedia aimed to create cohesive customer experiences across various client platforms while reducing complexity and accelerating innovation.

The adoption of Apollo Federation enabled Expedia to centralize their data and services into a single graph, facilitating collaboration between teams and providing a unified interface for accessing data. With a common entry point for clients, teams were freed from the requirement to directly connect to multiple services, resulting in improved efficiency and faster development cycles. The Apollo GraphOS platform enabled developers to seamlessly monitor, detect, and prevent schema breaks, ensuring the reliability and scalability of their graph infrastructure. This enabled developers to focus more on enhancing customer experiences rather than managing APIs, leading to a significant reduction in code complexity and technical debt.

By funneling all their digital capabilities into one central federated architecture powered by Apollo GraphOS, Expedia was able to condense several disparate tech stacks into one, build the trips experience 3x faster than their old approach and create a unified user experience across 25 brands.

Summary

In the digital era, organizations must embrace application modernization to remain competitive and deliver exceptional customer experiences. Apollo GraphOS emerges as a powerful solution, enabling businesses to accelerate their modernization journey while magnifying their existing API investments. By providing a unified yet modular architecture, Apollo GraphOS empowers organizations to increase developer productivity, reduce technical debt, and minimize operational costs and overhead. Apollo GraphOS not only supports present technological needs but also prepares organizations for future challenges, ensuring sustained competitive edge in an API-first world.

Connect with experts at Apollo today to learn how your organization can break the barriers to innovation and modernization with Apollo GraphOS.