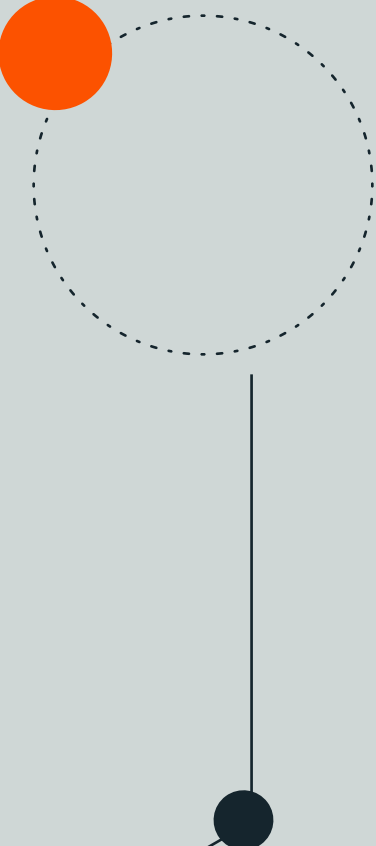




Platform Engineering for APIs: Using GraphQL to Drive Developer Efficiency

Table of contents

Introduction	01
Focus on API Developer Experience	02
Platform Engineering for API Management	04
Federated GraphQL: Crucial Layer of the Modern API Platforms	05
Apollo GraphOS For Automating GraphQL API Management	08
Safe and Rapid Graph Evolution	10
Transforming Developer Experience: GraphQL Success Stories	18
Conclusion	21
Reference List	22

A decorative graphic on the left side of the page. It consists of a solid orange circle at the top left, connected by a thin black line to a larger dashed black circle. A vertical black line extends downwards from the bottom of the dashed circle to a solid black circle. A diagonal black line then extends from the bottom-left of this solid black circle towards the bottom-left corner of the page.

Introduction

Platform engineering is emerging as a key force enabling organizations to deliver software faster and more reliably. A core focus of platform engineering is improving developer experience by creating an integrated internal developer platform that provides self-service, automation, standardization, centralized security, and composability capabilities. This paper explores how platform engineering teams at Netflix, Adobe, and others use GraphQL and Apollo Federation to standardize, automate, and scale API delivery.

You will learn how:

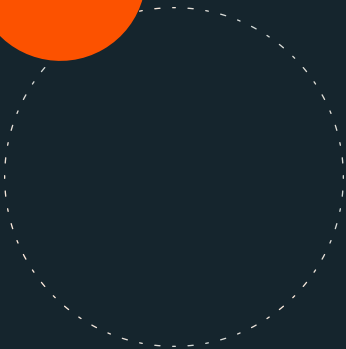
- A federated architecture unlocks GraphQL's benefits at scale for your organization.
- Apollo GraphOS improves developer experience by automating GraphQL API management.
- Leading organizations like Netflix, Booking.com and RetailMeNot are harnessing the power of GraphQL federation and Apollo GraphOS to boost developer productivity, improve reliability, and drive innovation.

Chapter 1

Focus on API Developer Experience

Today, Application programming interfaces (APIs) are the backbone of modern businesses with 83% of internet traffic consisting of API calls¹. APIs go beyond simply connecting data and applications; they empower businesses to curate personalized customer experiences, foster collaboration with external partners, and operate as robust engines for monetization².

Developers play a critical role at the center of the API economy. Developers are both the producers and consumers of APIs, responsible for managing the entire API lifecycle including design, build, test, integration, and API maintenance.





APIs dominate the developer workload. In fact, a recent Postman survey³ indicated that

“49% of respondents said most of their organization's development effort was spent working with APIs.”

Enabling faster and self-service access to these APIs isn't just helpful to developers – it contributes significant business value for an organization. McKinsey's recent research⁴ unveils a compelling correlation:

companies prioritizing developer velocity outperform their market counterparts by a staggering four to five times.

To win in this developer-centric world, platform engineering⁵ plays a crucial role in enhancing developer experience and reducing friction to release.



Platform Engineering for API Management

Platform engineering has emerged as a pivotal force in reshaping how organizations approach software development and delivery. Gartner predicts⁵ that by 2026,

approximately 80% of software engineering organizations will establish platform teams as internal providers of reusable services, components, and tools for application delivery.

A key aspect of platform engineering is a focus on the developer experience. This involves creating an integrated internal developer platform (IDP) that provides self-service, automation, standardization, centralized security, and composability.

While this practice isn't API-specific, it can contribute to our understanding of an effective approach to API lifecycle management. Platform engineering teams at leading organizations like Netflix⁶ and Adobe⁷ are turning to GraphQL⁸ and Apollo Federation⁹ for effective API management. These organizations are reaping substantial benefits by leveraging GraphQL as a composable abstraction layer atop their existing APIs, paving the way for accelerated application delivery and increased developer efficiency.

Federated GraphQL: Crucial Layer of the Modern API Platforms

When evaluating API architectures, GraphQL is commonly positioned as an alternative to REST. However, organizations are not replacing REST² or even SOAP in many cases. GraphQL is flexible enough to serve different purposes. With new GraphQL architectures like Apollo Federation, GraphQL can be leveraged as a complementary platform that sits on top of existing REST services, helping magnify an organization’s API investments.

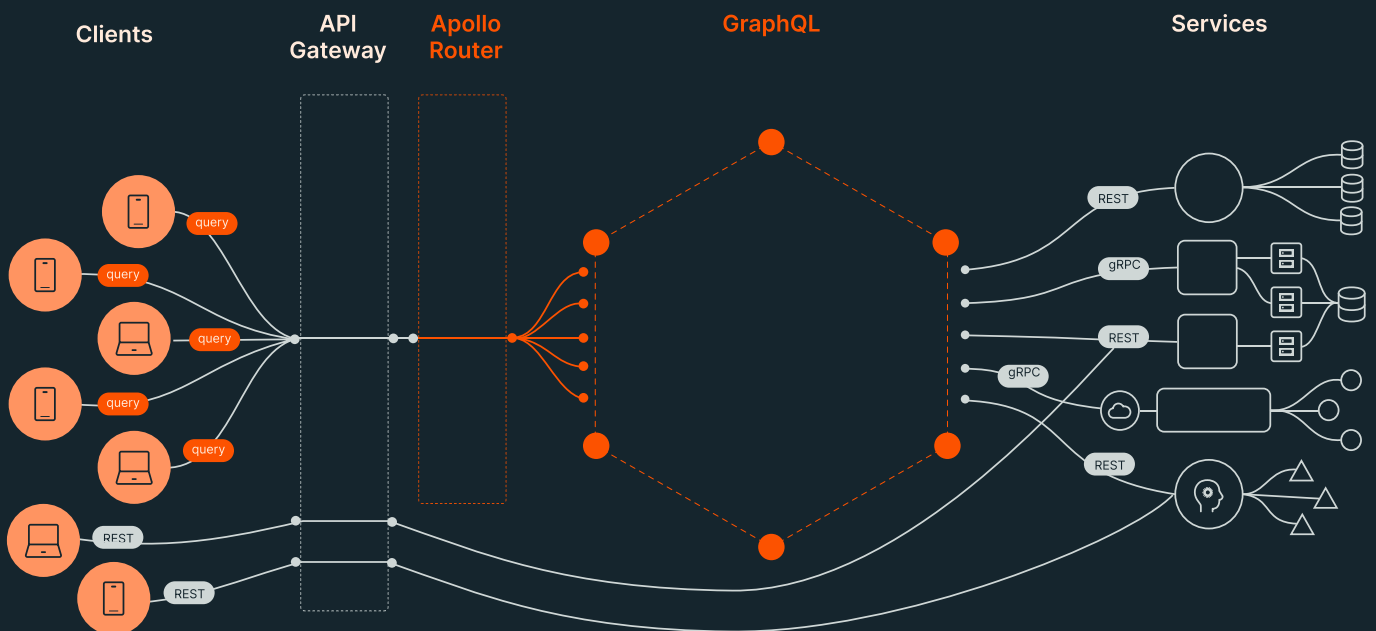


Figure: Federated GraphQL layer

When GraphQL is federated, it expands its capabilities beyond just a simple API in a stack.

Netflix⁶ highlights that a federated GraphQL platform

“solves many of the consistency and development velocity challenges with minimal tradeoffs on dimensions like scalability and operability.”

In a federated GraphQL architecture, any number of teams can contribute a GraphQL service to a unified graph. The graph is available to client teams via a single endpoint. This architecture provides the simplicity of a GraphQL monolith for client teams but the modularity of a more decoupled approach for service teams. The federated graph also handles API orchestration, ensuring the platform remains performant at all times. Teams can maintain individual GraphQL APIs, or subgraphs, that are accessed through a single router endpoint by clients. A composition process takes all subgraph schemas and intelligently combines them into a single schema, ensuring a consistent and performant runtime.

This supergraph architecture — a graph of graphs — enables service teams to support more clients with greater consistency and less redundant work. The federated approach helps strike a balance between decentralized ownership and integrated delivery, enabling organizations to scale GraphQL adoption in a governance-friendly manner.

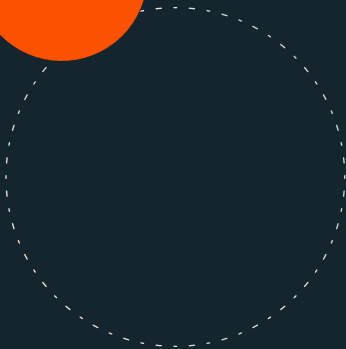
The GraphQL Advantage

While GraphQL is often associated with frontend development due to its client-centric approach, its true value extends far beyond the frontend. GraphQL serves as a powerful composition layer within an API platform, providing several benefits that align with the principles of cloud-native architecture¹⁰. Its ability to enable **flexible architecture, seamless developer experience, cost optimization, improved performance, along with a thriving open-source community**, makes GraphQL a crucial layer of the API platform. In the following table, we will examine these GraphQL benefits, exploring how they contribute to the agility, scalability, and adaptability crucial for modern cloud-native API ecosystems.

Dimension	Key Features of GraphQL
Architecture	<p>Decoupling: GraphQL ensures a clear contract between the frontend and the backend teams by defining a schema that outlines all available data types, their relationships, and the supported queries and mutations. This helps to decouple frontend and backend development. Frontend teams can work independently, designing queries that suit their UI requirements without direct dependencies on backend changes.</p> <p>Declarative and Hierarchical: GraphQL data is inherently declarative, enabling frontend teams to specify precisely the data they require. GraphQL facilitates complex data retrieval across multiple data sources in a single request, eliminating the need to stitch together multiple responses as often required in RESTful architectures.</p>
Developer Experience	<p>Strongly typed and introspection: GraphQL has strongly typed schema and introspection capabilities that enable early error detection via validation and improve developer productivity through auto-completion, API exploration, and documentation. This results in more stable APIs with fewer bugs in development.</p> <p>Versioning: GraphQL has backward compatibility with a built-in version control using the schema. This allows seamless incremental addition of new types to the schema without breaking changes to existing clients.</p>
Performance	<p>Prevents over-fetching and under-fetching: GraphQL empowers API clients to request precisely the data they need significantly reducing over-fetching and under-fetching issues commonly encountered in traditional RESTful APIs.</p> <p>Caching: GraphQL query responses can be cached granularly using directives¹ and cache-control headers, that optimizes resource usage by reducing server requests, enhancing performance, and delivering tailored, efficient data retrieval for varying client needs.</p>
Cost	<p>Efficient data transfer: GraphQL can aid in cost optimization by curbing unnecessary data transfer through its fine-grained data-fetching capabilities. It minimizes complexity and reduces network bandwidth consumption.</p> <p>Nested queries: GraphQL's ability to combine multiple requests into a single query can reduce server load, leading to more efficient resource utilization and potentially lower infrastructure costs.</p>
Ecosystem	<p>Community support & adoption: GraphQL benefits immensely from its vibrant open-source community. With over 62,000 GitHub stars and usage by major enterprises like GitHub, Netflix, PayPal, Expedia, Shopify, Twitter and others, GraphQL has seen rapid adoption. Companies like Apollo play a key role, actively building and contributing to the GraphQL ecosystem.</p>

Apollo GraphOS For Automating GraphQL API Management

As organizations discover the benefits of adopting a federated graph and its use spreads across different teams, the need for a scalable and evolvable graph solution becomes apparent. Developers want to tap into the power of graphs to build great products, without the burden of managing complex infrastructure themselves. They need a supergraph operating system that can scale globally to meet growing business needs, while letting them deploy rapid changes without breaking production.



The solution is Apollo GraphOS¹², the platform for building, managing, and scaling a supergraph. Apollo GraphOS provides a centralized registry and standardized workflows so that any team can contribute to the graph. It also provides centralized data plane to extend Apollo Router¹³, enabling organizations to accommodate their API security, scalability, and extensibility needs.

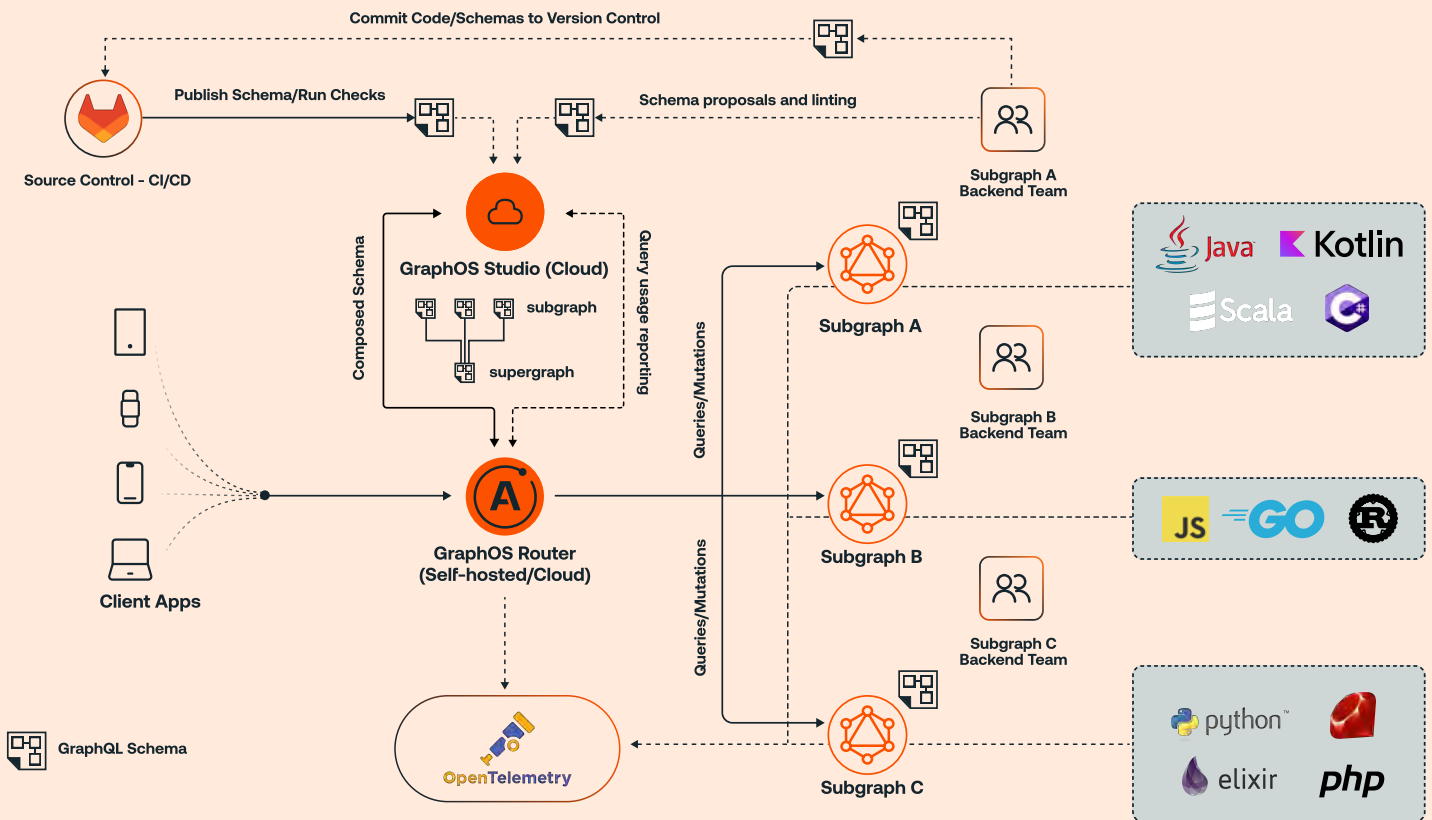
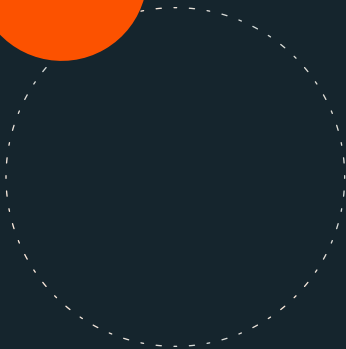


Figure: Supergraph Developer Tooling and CI/CD Pipelines

Safe and Rapid Graph Evolution

Apollo GraphOS provides everything you need to build an API platform for the modern stack. It empowers teams to adopt a DevOps approach for deploying and managing GraphQL schemas and APIs through a series of robust features. Apollo GraphOS gives development teams the tools to develop schemas collaboratively with a single source of truth, deliver changes safely with graph CI/CD, and improve performance with field and operation-level observability.

This section outlines best practices for leveraging Apollo GraphOS at each DevOps stage to continuously deliver value on graphs with speed and safety, ultimately boosting the developer experience.



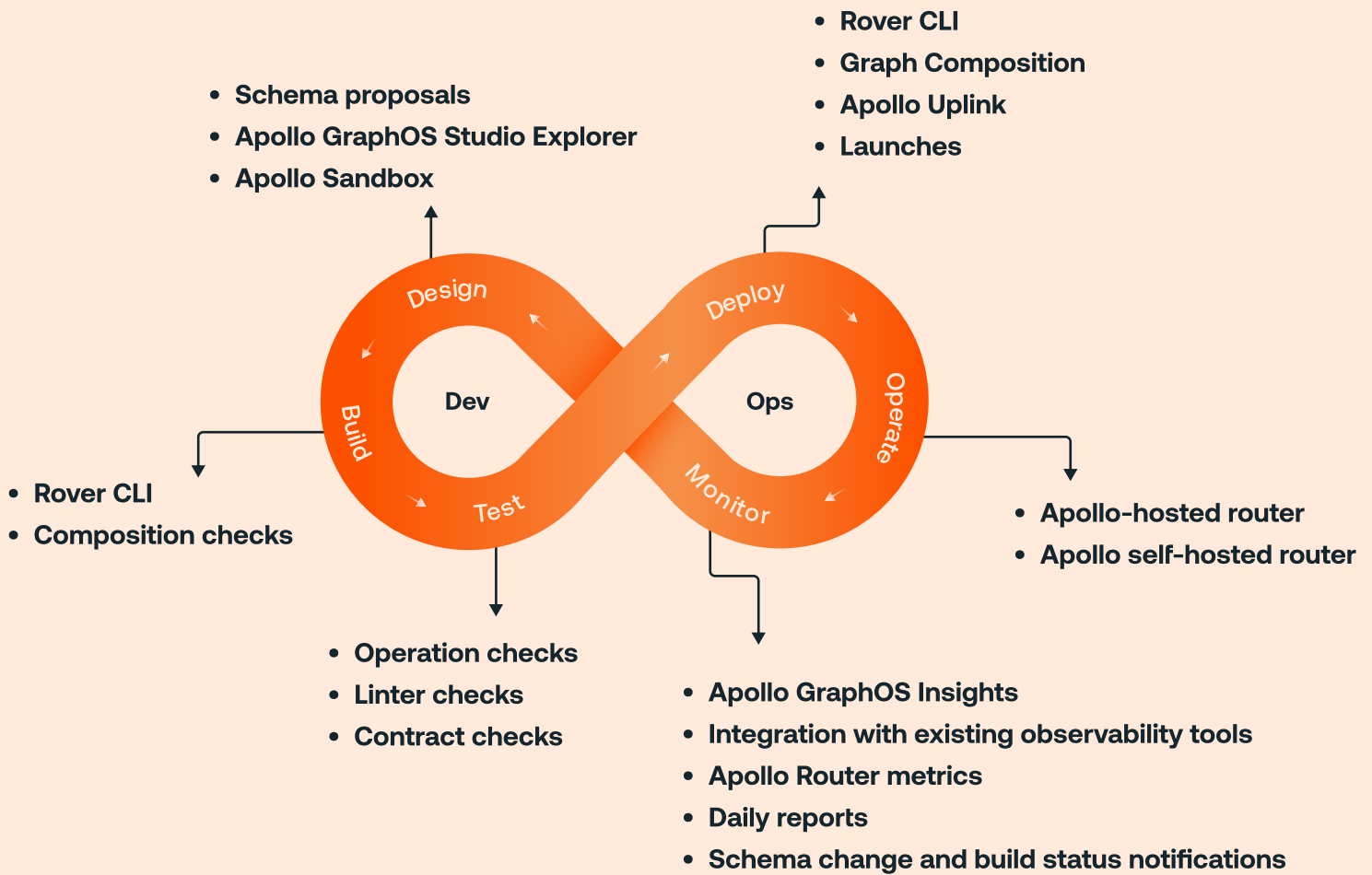


Figure: Applying DevOps phases to GraphQL API management

Design

The first phase in GraphQL API lifecycle is the creation and structuring of GraphQL schemas. These schemas dictate how clients interact and retrieve data from the GraphQL API. The schema's power lies in its flexibility: it shouldn't be rigidly tied to specific service implementations or clients. Embracing demand-oriented schema¹⁴ design, especially with specifications like Apollo Federation, ensures schemas evolve flexibly to support diverse client requirements while avoiding overemphasis on any single client's needs.

As organizations expand and innovate, their GraphQL schemas undergo continuous growth and evolution. The addition of new products and features leads to the introduction of new schema types and fields. Implementing any such updates to a subgraph's schema requires clear cross-team communication to comprehensively understand, verify, and track changes, mitigating the risk of any breaking changes that might disrupt existing functionalities.

Apollo GraphOS’s schema proposals¹⁵ offer a solution by providing a centralized process for proposing, validating, and implementing schema changes, ensuring efficient and collaborative development cycles with governance controls.

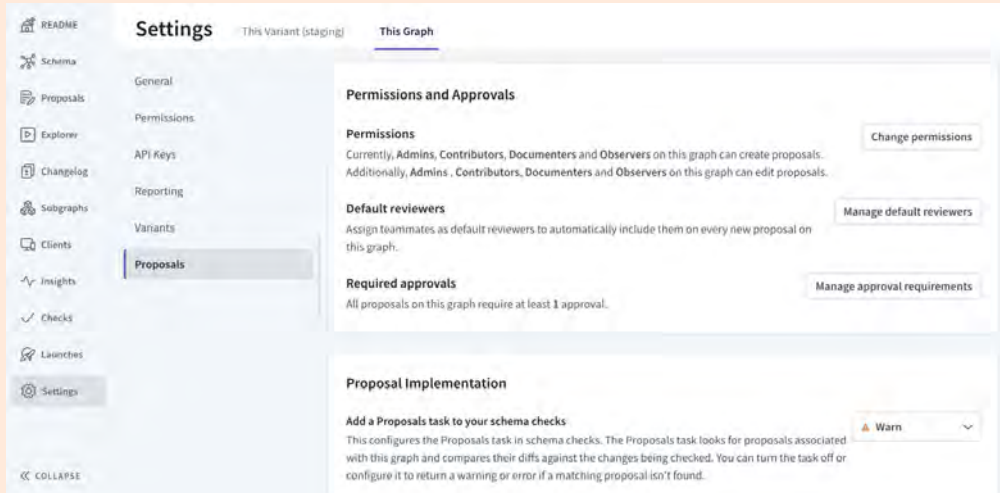


Figure: Schema proposals in GraphOS Studio

Additionally, Apollo offers various tools designed to expedite API development. Apollo GraphOS Studio Explorer¹⁶, a powerful web IDE, enables developers to seamlessly create, run, and manage GraphQL operations: Query, Mutation, and Subscription. While developers are making local changes to an individual subgraph, they can use the `rover dev` command to start a local router instance and get a local Apollo Studio Explorer instance.

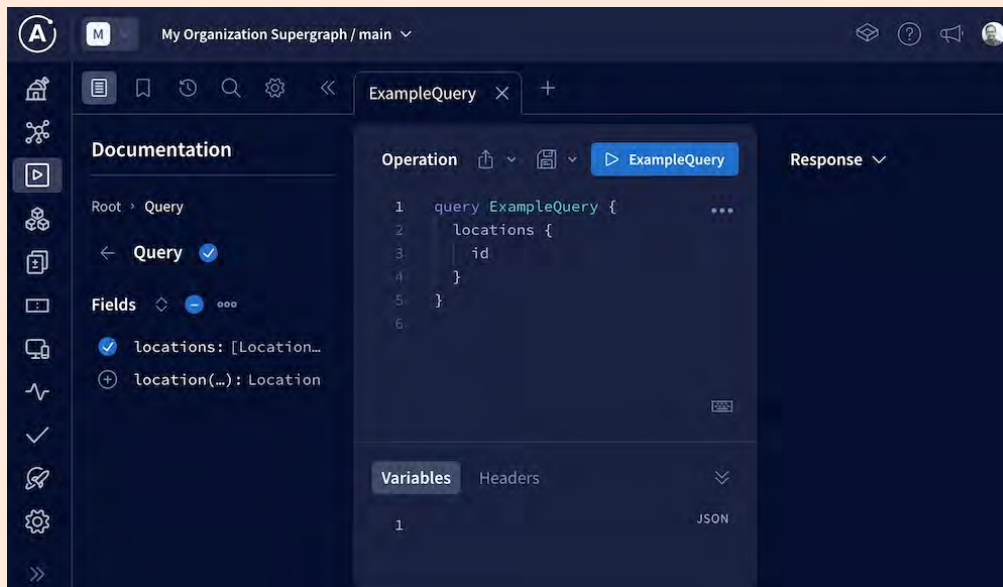


Figure: The GraphOS Studio Explorer

Build

In the build phase of the GraphQL API lifecycle, the focus lies on ensuring the successful composition and integration of the subgraph schema.

Once the subgraph schema changes are approved, composition checks¹⁷ are triggered locally to verify whether the proposed schema changes will successfully compose with other subgraph schemas. A composition check verifies that changes to a subgraph schema are valid GraphQL definitions and are compatible with the other subgraph schemas, enabling them to compose into a supergraph schema for the router. Composition checks can be run within existing CI/CD tools using Apollo's Rover CLI¹⁸ to create a seamless integration with existing software delivery pipelines.

If composition succeeds, a series of other schema checks are triggered for further validation. If composition fails, validation ends and results are returned to the developer.

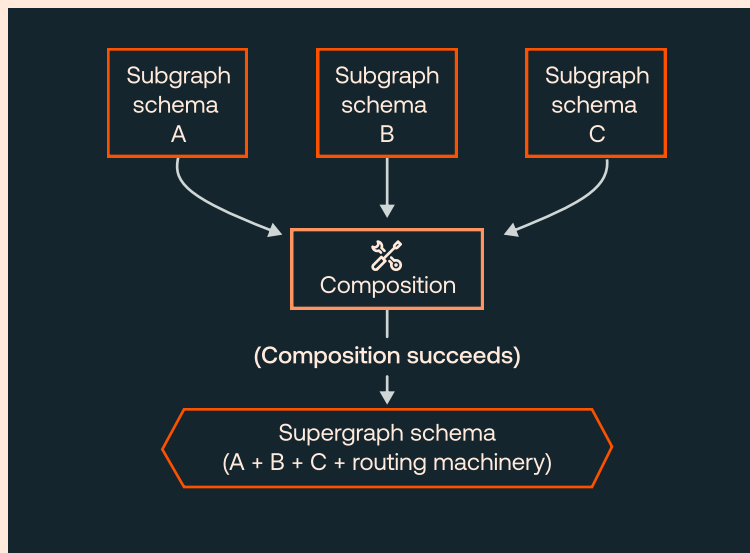


Figure: Composition checks

Test

During the test phase, Apollo GraphOS ensures proposed schema changes follow best practices to ensure the robustness and reliability of the GraphQL infrastructure. It provides crucial schema checks¹⁹, a schema governance tool, to prevent breaking changes before implementation. Just like composition checks, schema checks can be run locally within existing CI/CD tools using the Rover CLI creating a seamless integration with existing software delivery pipelines.

Apollo GraphOS can perform the following types of schema checks:

- **Operation checks** - If a composition check succeeds, Apollo GraphOS then validates schema changes with operation checks. It compares the proposed schema changes against historical operations to verify whether the changes will break any of the graph's active clients.
- **Linters checks** - Apollo GraphOS provides schema linting that help analyze proposed schema changes for violations of formatting rules and other GraphQL best practices.
- **Contract checks** - Apollo GraphOS contracts enable developers to deliver different subsets of their supergraph to different consumers. Contract checks help validate that proposed schema changes won't break any downstream contract variants.

If any of these checks fail, developers will see the errors in the GitHub PR, which will also link to GraphOS Studio for more details. Developers will need to investigate the error, fix it in their local environment and follow the validation process again from the start. If all the checks pass, the schema changes are ready to be composed into the supergraph schema so that they are available to the API clients.

Deploy

Once the proposed schema changes pass all the schema checks, they can be deployed in the Apollo GraphOS platform using the Rover CLI or GraphOS Platform API²⁰. The platform stores the updated schema(s) in its schema registry. At its core, the schema registry is a version control system for the schema. It stores schema's change history, tracking the types and fields that were added, modified, and removed. When the schema registry gets a new or updated version of a subgraph schema, it validates²¹ and composes²² it into a supergraph schema.

The schema registry automatically sends the supergraph schema to an internal service within Apollo GraphOS called Apollo Uplink²³. Uplink is a server that stores the latest supergraph schema for each graph. The router fetches the latest schema from Uplink and uses this new schema to respond to client requests.

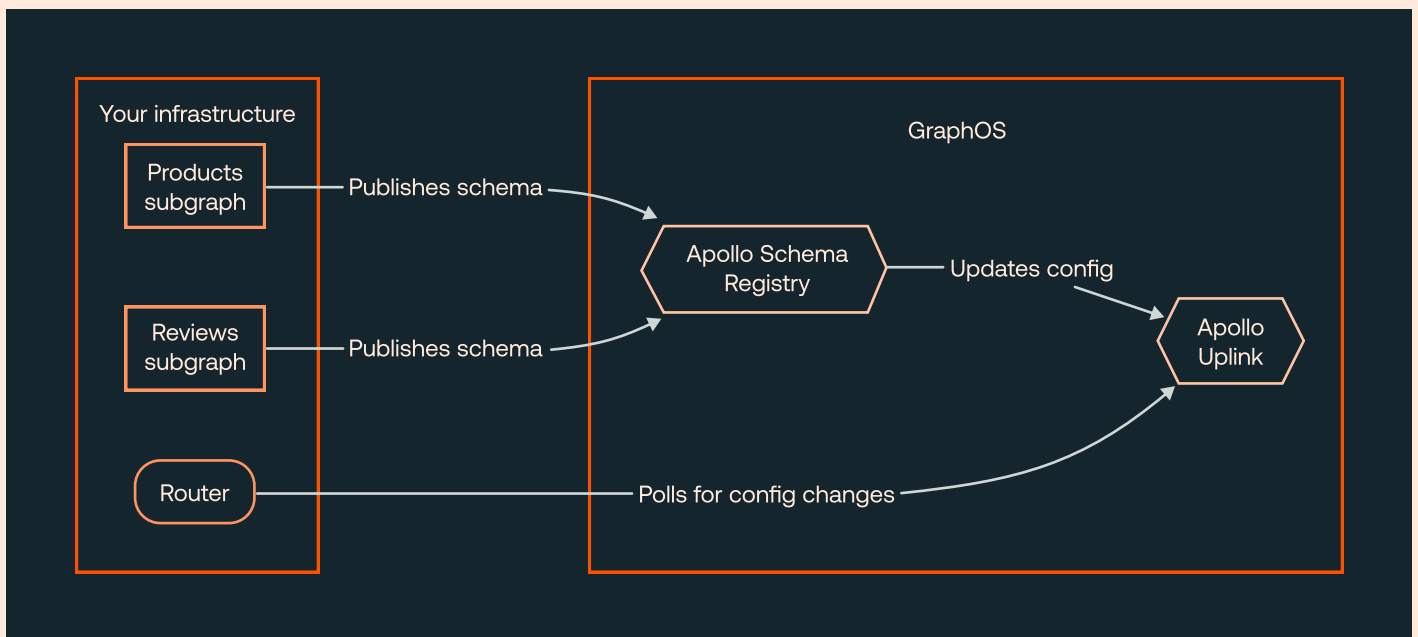


Figure: Schema publishing process

The Launches²⁴ page in GraphOS Studio enables developers to observe and monitor the schema delivery process for both in-progress and past launches. It's a dashboard that summarizes how launches over time have evolved the supergraph, helping developers track when new changes are reflected in production and debug when something goes wrong.

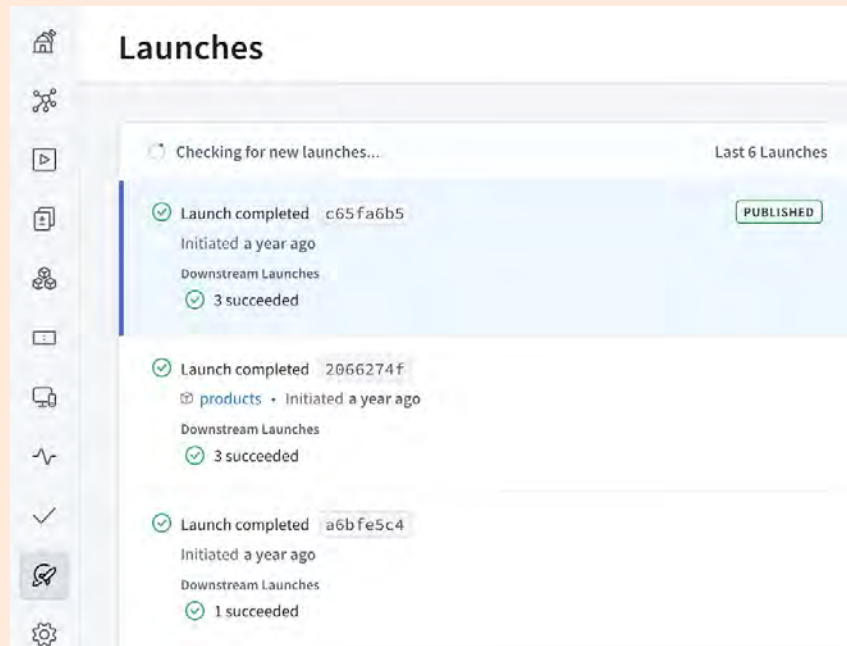


Figure: Launches page in GraphOS Studio

Operate

Once the federated graph is deployed, it must be managed on an ongoing basis. The Apollo Router, integrated within the Apollo GraphOS platform, plays a pivotal role in facilitating the ongoing management of a deployed GraphQL API. Users have the flexibility to operate Apollo Router seamlessly either in Apollo-hosted²⁵ or self-hosted²⁶ environments.

The router acts as the entry point to the subgraphs, providing a unified interface for clients. It intelligently executes each client operation across the appropriate subgraphs and merges the responses into a single result for the client. The router offers comprehensive observability, allowing organizations to track usage down to individual fields and operations.

Monitor

The final phase of the DevOps cycle is to monitor the environment by collecting data and providing analytics on the graph's behavior, performance, errors and more.

Apollo GraphOS Studio²⁷ offers a performant and intuitive user interface to help monitor and understand supergraph's usage and performance. Users can seamlessly integrate with their existing tools like Datadog to monitor their graph's performance.

As discussed earlier, the router is the single entry point to the subgraphs, all client requests pass through the router's request lifecycle. Therefore, the router's observability is critical for maintaining a healthy, performant supergraph and minimizing its mean time to repair (MTTR). The Apollo Router provides the necessary telemetry, logging and tracing data to monitor its health and troubleshoot issues. Users can also configure Apollo Router to export router and graph traces and metrics to their choice of observability tools using OpenTelemetry. Additionally, customizing instrumentation and events can be done simply by configuring the Apollo Router.

In addition to the observability around graph performance, Apollo GraphOS Studio also provides daily reports of graphs' activity, schema change and build status notifications to empower teams with real-time insights, enabling them to maintain, optimize, and innovate their graph infrastructure effectively.

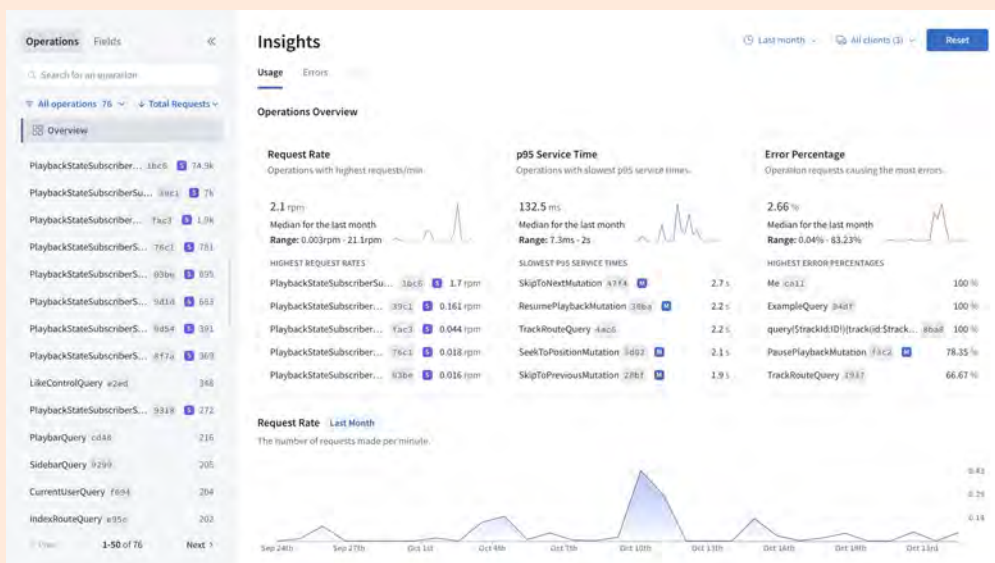


Figure: GraphOS Insights

Chapter 06

Transforming Developer Experience: GraphQL Success Stories



NETFLIX

Netflix⁶, renowned for its microservice architecture, faced challenges with its UI developers managing hundreds of microservices and backend developers grappling with complexity and resilience issues. They introduced a unified API aggregation layer to address this, but as their business and developer count grew, this layer became increasingly challenging to maintain. Netflix adopted GraphQL federation, leveraging Apollo Federation Specification, to resolve this. A federated GraphQL platform solved many of the consistency and development velocity challenges while maintaining scalability and operability. This approach promised benefits of a unified API schema for consumers while also giving backend developers flexibility and service isolation they required.

Booking.com

Booking.com, a global leader in travel accommodations, faced challenges stemming from an aging monolithic architecture that led to inconsistent experiences across clients and increased maintenance costs. The Booking.com team turned to Apollo GraphOS platform²⁸ to decouple frontend and backend development and gain comprehensive visibility into API consumption. With Apollo's platform, they swiftly transitioned to a modern, distributed architecture, empowering teams with enhanced autonomy and streamlining feature deployment. The implementation of schema checks fortified resilience and reliability, allowing developers to innovate without the fear of breaking systems. Booking.com achieved a remarkable 40% acceleration in shipping speed by adopting federated approach, complemented by seamless automation and integrated CI/CD processes.

“We are already seeing benefits of our supergraph preventing mistakes when people push a bad schema. It's stopping silly mistakes before they happen, giving developers confidence. It's really started helping the developers move, and they're much happier with the experience since they can build without breaking things.”

- Matt Sexton, Solutions Architect, Booking.com

RetailMeNot.inc.

RetailMeNot, a leading savings destination that brings shoppers incredible offers, revolutionized its technology stack in 2019 to scale and enhance user experiences²⁹. Starting with REST APIs, they shifted to a monolithic GraphQL API, but growing contributions turned into a bottleneck. Recognizing the need for scalability, they turned to Apollo Federation, empowering discrete subgraph teams to maintain portions of a unified supergraph schema, curbing inconsistencies and streamlining data models. Incrementally migrating from the monolith, they optimized their schema and traffic routing, witnessing an immediate shift post-Apollo Federation adoption. Automation through Apollo Studio reduced code review time by over 65%, enabling teams to innovate, educate on GraphQL best practices, and fortify their supergraph. This transformation eliminated all breaking changes, bolstered reliability, and boosted feature delivery by 40%, marking a confident shift from monolithic constraints to agile subgraph architecture.

After adopting managed federation and schema checks, we went from three engineers spending 75% of their time reviewing code to less than 10%.”

- Hannah Shin, Senior Software Engineer, RetailMeNot

Conclusion

GraphQL and Apollo GraphOS provide a strategic pathway to build a resilient, developer-friendly API platform. By implementing GraphQL in platform engineering efforts, organizations can modernize their APIs and provide delightful user experiences. The Apollo GraphOS platform further extends GraphQL's capabilities by automating federated schema management, integrating CI/CD workflows, and providing robust observability.

By adopting GraphQL and Apollo GraphOS, platform engineering teams can deliver the self-service, automation, standardization, and centralized governance needed for delivering stellar developer experiences. Leading companies like Netflix, Booking.com, and RetailMeNot have used these technologies to boost developer velocity, reliability, and business agility.

Ready to embark on this journey? Learn how your organization can modernize your API platform with GraphQL by [consulting the experts at Apollo](#).

Further Reading

¹ “Akamai State of the Internet Security Report.”, 26 February 2019, <https://www.akamai.com/newsroom/press-release/state-of-the-internet-security-retail-attacks-and-api-traffic>.

² Gilling, Derric. “Turning APIs Into Revenue Centers By Monetizing Usage.”, 13 October, 2023, <https://www.forbes.com/sites/forbestechcouncil/2023/10/13/turning-apis-into-revenue-centers-by-monetizing-usage/?sh=1947482460c9>.

³ “2023 State of the API Report.”, 2023, <https://www.postman.com/state-of-api/a-day-week-or-year-in-the-life/#api-development-effort>

⁴ Srivastava et al., “Developer Velocity: How software excellence fuels business performance”, 20 April, 2020, <https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/developer-velocity-how-software-excellence-fuels-business-performance>

⁵ Perri, Lori. “What Is Platform Engineering?”, October 26, 2023, <https://www.gartner.com/en/articles/what-is-platform-engineering>

⁶ Netflix Technology Blog. “How Netflix Scales its API with GraphQL Federation (Part 1)”, 9 November, 2020, <https://netflixtechblog.com/how-netflix-scales-its-api-with-graphql-federation-part-1-ae3557c187e2>

⁷ Anderson & Ross. “GraphQL: Making Sense of Enterprise Microservices for the UI”, 4, March, 2021, <https://blog.developer.adobe.com/graphql-making-sense-of-enterprise-microservices-for-the-ui-46fc8f5a5301>

⁸ “Why GraphQL”, <https://www.apollographql.com/why-graphql>

⁹ “Introduction to Apollo Federation”, <https://www.apollographql.com/docs/federation/>

¹⁰ Grey, Tom. “5 principles for cloud-native architecture—what it is and how to master it.”, 19 June, 2019, <https://cloud.google.com/blog/products/application-development/5-principles-for-cloud-native-architecture-what-it-is-and-how-to-master-it>

¹¹ GraphQL.org, <https://graphql.org/learn/queries/#directives>

¹² Apollo GraphQL docs, “The Apollo GraphOS platform.”, <https://www.apollographql.com/docs/intro/platform>

¹³ Apollo GraphQL docs, “The Apollo Router.”, <https://www.apollographql.com/docs/router/>

¹⁴ Apollo GraphQL docs, “Demand oriented schema design”, <https://www.apollographql.com/docs/technotes/TN0027-demand-oriented-schema-design/>

¹⁵ Apollo GraphQL docs, “Schema proposals”, <https://www.apollographql.com/docs/graphos/delivery/schema-proposals>

¹⁶ Apollo GraphQL docs, “The GraphOS Studio Explorer”, <https://www.apollographql.com/docs/graphos/explorer/>

¹⁷ Apollo GraphQL docs, “Composition checks”, <https://www.apollographql.com/docs/graphos/delivery/schema-checks/#composition-checks>

¹⁸ Apollo GraphQL docs, “The Rover CLI”, <https://www.apollographql.com/docs/rover/>

¹⁹ Apollo GraphQL docs, “Schema checks.” <https://www.apollographql.com/docs/graphos/delivery/schema-checks/>

²⁰ Apollo GraphQL docs, “The GraphOS Platform API.” <https://www.apollographql.com/docs/graphos/platform-api/>

²¹ Apollo GraphQL docs, “Federated schema checks.” <https://www.apollographql.com/docs/federation/managed-federation/federated-schema-checks/>

²² Apollo GraphQL docs, “Schema checks.”, <https://www.apollographql.com/docs/federation/federated-types/composition/>

²³ Apollo GraphQL docs, “Apollo Uplink.”, <https://www.apollographql.com/docs/federation/managed-federation/uplink/>

²⁴ Apollo GraphQL docs, “Launches.” <https://www.apollographql.com/docs/graphos/delivery/launches/>

²⁵ Apollo GraphQL docs, “Cloud routing for supergraphs.”, <https://www.apollographql.com/docs/graphos/routing/cloud/>

²⁶ Apollo GraphQL docs, “Set up a self-hosted supergraph.”, <https://www.apollographql.com/docs/graphos/quickstart/self-hosted/>

²⁷ Apollo GraphQL docs, “Metrics and insights in GraphOS.”, <https://www.apollographql.com/docs/graphos/metrics/>

²⁸ “The supergraph helps Booking.com boost developer productivity and ship 40% faster.”, <https://www.apollographql.com/customers/booking>

²⁹ “RetailMeNot Ships 40% Faster and Eliminates Downtime with their Supergraph.”, <https://www.apollographql.com/customers/retailmenot>