# **How GraphQL turned web development on its head**

**[Matt Asay](https://www.infoworld.com/author/Matt-Asay/) Contributor, InfoWorld**

Web giants like Facebook and Google are setting a torrid pace to make life better for front-end developers. By creating new technologies like React, supported by frameworks like ​Vercel’s [Next.js](https://nextjs.org/)​ and [Netlify](https://www.netlify.com/), these companies free developers to focus on the logic of their new applications and services while abstracting away the back-end issues that have historically slowed developers.

Facebook rightly gets credit for its front-end work with open source React, but arguably it may deserve even more credit for open sourcing an emerging query language called [GraphQL](https://www.infoworld.com/article/3269074/what-is-graphql-better-apis-by-design.html). In a new cloud native world of microservices, the number of endpoint APIs increases exponentially, which is proving a huge challenge for traditional SQL technologies.

GraphQL, however, can communicate across microservices and operate almost as a gateway to all of your services — a unifying source that provides a holistic view of all your systems. Turns out that’s a Very Good ThingTM.

To understand what is behind the meteoric rise of GraphQL, I spoke to Manish R. Jain, founder and CEO of ​[Dgraph Labs](https://dgraph.io/)​. His team created a database around GraphQL from scratch in Go, all available as open source.

**Developer darling**

Released as open source in 2015 and ​now [under the guidance of the Linux Foundation](https://www.infoworld.com/article/3318255/graphql-gets-its-own-foundation.html), GraphQL is behind a surprising number of prominent websites including Facebook, Google, Airbnb, Pinterest, and more ([current list​ ​here​](https://graphql.org/users/)). A 2019 survey of more than 20,000 JavaScript developers found​ ​[GraphQL adoption exploding](https://blog.graphqleditor.com/the-state-of-graphql/)​. A year after its release, 36% of developers surveyed had never heard of GraphQL. By 2017 that number was cut in half to 17.9%. Today it’s 7.1%, with 62.5% of survey respondents saying they want to learn GraphQL.

Nor is GraphQL reserved for high-flying Internet companies. For example, Netflix recently shared its success with GraphQL on its​ ​[developer blog](https://netflixtechblog.com/our-learnings-from-adopting-graphql-f099de39ae5f)​, finding that GraphQL on Node.js proved to “significantly increase our development velocity and overall page load performance.”

All of which suggests that this GraphQL thing sort of rocks.

Not a query language?

​That said, GraphQL might be poorly named. Not the graph part. That’s accurate. But the “query language” appendage may not be so apt.

At least, that’s what Jain believes, and he may be in a good position to judge. Before founding Dgraph Labs, Jain helped build graph database technologies at Google. He led the project to unite all Google OneBoxes under a graph indexing system, which involved weather, flights, events, and so on (​e.g., search for “[weather in SF](https://www.google.com/search?q=weather+in+sf)​”). He later left Google and subsequently founded Dgraph Labs with backing from Redpoint Ventures.

“GraphQL is one of the bets we took very early on in ​[Dgraph​’s](https://github.com/dgraph-io/dgraph) journey,” Jain said. “When looking for a query language for the database back in 2015, we weren’t happy with the existing choices of Cypher and Gremlin. Incidentally, Facebook’s GraphQL draft spec had just been publicly ​released​ in July 2015. We ​really​ liked it​. ​We bought the simplicity and power of this system, but also felt like we were taking a risk in going with unproven technology, particularly considering it is a syntax for describing an API ([like REST​](https://www.infoworld.com/article/3269074/what-is-graphql-better-apis-by-design.html)) rather than a query language.​”

Back to that “QL” quibble. Jain wants to make a key point here: ​GraphQL is not a graph database query language; rather, it’s a replacement for REST APIs. How’s that?

**A replacement for REST**

GraphQL is designed to build client applications by providing an intuitive and flexible syntax for describing data requirements. In more approachable vernacular, Jain explained, GraphQL allows a way for clients to understand what the server provides and specifically request data useful to the client. ​It provides a simple way for developers to retrieve only as much of their [​strongly-typed dataset](https://en.wikipedia.org/wiki/Strong_and_weak_typing)​ as they need, and in the shape in which they need it.

In a ​[blog post](https://dgraph.io/blog/post/building-native-graphql-database-dgraph/)​, Jain explained in more technical detail:

REST API endpoints are fixed. Each endpoint would give you a predetermined amount of data, irrespective of how much data the client needs. Squeezing efficiency in such a system leads developers to generate an ever-increasing number of endpoints, each returning a contained set of results. The client would then call them in sequence to generate a view for the end-user. There are guidelines around what would be an HTTP POST, what would make a PUT, what would be a GET, and what the return error codes would be.

GraphQL turns all of this on its head. There’s only one endpoint, most ​accesses are POST, all return codes are HTTP 200, with the errors captured in a JSON response. The server publishes what all the clients can access, and the client determines how much it wants to access, depending upon its use case. It is simple and powerful. Considering GraphQL is put in the same category as ​REST, gRPC, and webhooks​, calling it a ​‘QL’​ (Query Language) is a bit of a stretch.​ ​It is really a syntax for describing an API​.​ If you take up GraphQL, you get to build whatever API you like and use GraphQL to describe the API’s types and operations. Those things then describe the data that your API can return.

Which brings us back to Dgraph Labs. According to Jain, “Dgraph is unique in that it is the only native GraphQL database. We designed a database that stores and distributes the data in a way most optimal for executing GraphQL traversals, joins, and retrievals.”

In Jain’s view, with the right technology, most codebases can be cut down significantly. “We want to enable developers in building faster, smarter applications, spinning up performant, scalable back ends instantly.”