Editor: **Robert Blumen**
Katana Graph
robert@robertblumen.com

APIs With GraphQL

Jeff Doolittle

From the Editor

In Episode 530 of "Software Engineering Radio," Tanmai Gopal, CEO of Hasura.io, discusses GraphQL with SE Radio host Jeff Doolittle. The topics include the history and rationale behind GraphQL, use cases for which it is suited, how GraphQL differs from other application programming interface specification styles such as REST and gRPC, and GraphQL performance, caching, and security. Summary excerpts are provided later in this article. To hear the full interview, visit http://www.se-radio.net, or access our archives via RSS at http://feeds.feedburner.com/se-radio.—*Robert Blumen*

Jeff Doolittle: What is GraphQL?

Tanmai Gopal: GraphQL is an API (application programming interface) specification, similar to other API specifications like REST and SQL to some degree. It was originally developed at Facebook before they open sourced it around 2016. It was intended to sit at the layer between front-end and back-end applications, and most people use it as a better API specification in their applications for their back-end systems. GraphQL reduces endpoint proliferation in API fetch calls, problems with inconsistent documentation, and difficulties of communication between front-end and back-end teams.

JSON (JavaScript Object Notation) has emerged among developers as the de facto communication standard for exchanging data between decoupled systems; GraphQL makes it convenient to query a set of endpoints that respond with JSON.

How does GraphQL support subscriptions?

Subscriptions are queries of events, real-time information that is typically coming in over WebSockets. Subscriptions are a bit like pub/sub messaging. You're looking at data that comes in from the server and is sent to the client.

Implementing this with REST, there's a lot of variation possible: how to set up a WebSocket client or set up a WebSocket server, the protocol for authentication/authorization, and how to exchange information. With GraphQL subscriptions you still use WebSockets, but because there's a graphical specification, the consumption of these real-time events or data becomes easy on the client side, and because you're just using a normal GraphQL client, it becomes a nicer integration experience. You still have to do the hard work of having a GraphQL server that can actually fetch data from a stream and do the authorization work, but at least from a protocol point of view, for the client and server setup, this is standardized, so you have less to worry about.

Digital Object Identifier 10.1109/MS.2022.3227254
Date of current version: 13 February 2023

0740-7459/23©2023IEEE

What would you put a GraphQL API on top of?

You can put a GraphQL API on top of anything if you're willing to do the work of building the code that cannot just process the query but also understand how to go fetch the data from various sources.

You can build your own GraphQL API, and you can also put a GraphQL API on nothing; it can literally just create some data and send it to you for testing the API service. You can have a GraphQL API that ultimately goes and talks to a Postgres or any relational database, and you can put it on top of a NoSQL database or a graph database. Anything that's actually a back-end store of data.

Developers are increasingly building applications at the edge. The edge is this evolution of mobile web serverless workers who write things that happen in a different security context and a different performance context from the rest of your centralized data and logic. It's the decentralized data and logic-processing part of your business. You can give those developers access to stuff over GraphQL, and they love you for it. In scenarios where people want to consume these APIs, especially at the edge, GraphQL can expose that data. And then you can decide what to stick GraphQL on top of.

How do I deal with performance and caching in GraphQL?

GraphQL flipped control of what data are being fetched to the client. In REST, you as the API developer decide what data your consumer will get. In GraphQL, the GraphQL client decides what data they want to fetch. This is the root cause of all tradeoffs. All of the cons and complexities of GraphQL, as well as all of the pros, come from the client being able to decide what they want. As soon as you let the client decide, you might have inefficient ways of fetching that data because you don't know what data you want up front. That means that in this back-end system that I'm building, I have to be able to fetch the data sources that I have as optimally as possible and create an optimal query plan and data-fetching route.

On the caching front it's similar, where the existing caching infrastructure breaks down when it comes to GraphQL. But it's one of the things that people are building and figuring out, having different approaches to server-side caching.

How does GraphQL approach the security implications of who can see what?

The short answer is, GraphQL doesn't care. It says, "You do security. We give you an API specification that lets you define how you want to fetch data, how you're mutating stuff, how you're subscribing to stuff; then you do security the way you need to."

What tooling is available in the GraphQL ecosystem?

It's similar to what you would expect in an API ecosystem. You have client side, server side, and collaboration or SDLC (software development lifecycle) tooling. Popular GraphQL clients

SOFTWARE ENGINEERING RADIO

Visit www.se-radio.net to listen to these and other insightful hour-long podcasts.

RECENT EPISODES

- 542—Brendan Callum, engineering manager for the Pinterest developer platform team, and host Kanchan Shringi, discuss the "spec-first" approach to API development and how it's different from "API first."
- 541—Open source developers Jordan Harband and Donald Fischer join host Robert Blumen for a conversation about securing the software supply chain, especially open source.
- 539—Adam Dymitruk, CEO and founder of the Adaptech Group, joins host Jeff Doolittle for an exploration of the event modeling approach to discovering requirements and designing software systems.

UPCOMING EPISODES

- Host Priyanka Raghavan talks to Ganesh Detta regarding DevOps versus SRE.
- John deVadoss discusses .NET and Azure design patterns with host Nikhil Krishna.
- Host Kanchan Shringi talks to Nicholas Manson about identity management for cloud application.

integrate well with your application-development environment when you're writing to web, mobile, or a Java service. Different languages, devices, and platforms have clients that can autogenerate the SDK (software development kit). While you're building a web application or something that talks to the GraphQL API, you can do an SDK generator and call the CLI command that generates the SDK on demand based on the GraphQL schema and agenda.

Tools on the server side help you build GraphQL services. This is the work we're doing in Hasura. There are also GraphQL-as-a-service kinds of tools and GraphQL server tools that help you build a GraphQL service from scratch.

Then there are API collaboration tools for GraphQL. If your GraphQL schema is changing, you want to see how it changes. You make commits to your Git repository, and you want to trigger certain things that verify whether the GraphQL schema has evolved in a way that might break the schema, like removing the field from the user. And it will do an automatic check.

The GraphQL ecosystem tooling on the monitoring side has been slower to catch up. That category of tools for monitoring and observability on the caching side is starting to grow gradually. You'll see these tools integrated with the community members or vendors who are building things with the GraphQL server side. So, the GraphQL server tools try to also solve for some of those operational problems. We hope over time to see that separate out a little more, with things like GraphQL-specific Open Telemetry.

ABOUT THE AUTHOR

JEFF DOOLITTLE, senior software architect at Trimble Viewpoint in Portland, CA 97214 USA, is a master architect with over 25 years of industry experience. Contact him at jeff@jeffdoolittle.com or at jeffdoolittle.com.

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscribe to *IEEE Software* by visiting www.computer.org/software.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.