EPISODE 1022

[INTRODUCTION]

[00:00:00] JM: GraphQL is a system that allows frontend engineers to make requests across multiple data sources using a simple query format. In GraphQL, a frontend developer does not need to worry about the request logic for individual backend services. The frontend developer only needs to know how to issue GraphQL requests from the client and these requests are handled by a GraphQL server.

GraphQL is mostly used to issue queries across internal databases and services, but many of the data sources that a company needs to query in modern infrastructure are not databases. They're APIs like Salesforce, Zendesk or Stripe. These API companies might store a large percentage of the data that a given company needs to query, and executing queries and subscriptions and joins against these APIs is not a simple task.

OneGraph is a company that builds integrations with third-party services and exposes them through a GraphQL interface. Sean Grove is a founder of OneGraph and he joins the show to explain the problem that OneGraph solves, how OneGraph is built and some of the difficult engineering challenges required to design OneGraph.

[SPONSOR MESSAGE]

[00:01:22] JM: Today's show is sponsored by Datadog, a scalable, full-stack monitoring platform. Datadog synthetic API tests help you detect and debug user-facing issues in critical endpoints and application. Build and deploy self-maintaining browser tests to simulate user journeys from global locations. If a test fails, get more context by inspecting a waterfall visualization or pivoting to related sources of data for troubleshooting. Plus, Datadog's browser tests automatically update to reflect changes in your UI so you can spend less time fixing tests and more time building features.

You can proactively monitor user experiences today with a free 14-day trial of Datadog and you will get a free t-shirt. Go to softwareengineeringdaily.com/datadog to get that free t-shirt and try out Datadog's monitoring solutions today.

[INTERVIEW]

[00:02:31] JM: Sean Grove, welcome to Software Engineering Daily.

[00:02:33] SG: Hey! Thank you for having me.

[00:02:34] JM: GraphQL was released in 2015. When did you start working with GraphQL?

[00:02:40] SG: I think I built the first prototype of a tool with it in about January or February of 2015 probably.

[00:02:46] JM: And what were your reactions to it when you started working with it?

[00:02:50] SG: At the time, I was building an extension for Gmail, and the idea was you open up an email and it would show you information on the person you're talking to from your Stripe account, Mailchimp, about 16 in different services. Kind of like reportive, but for the data. There was obviously like this tree of data that kind of expand all these different services and it was hellish to go in to all the services to pull out the data I wanted.

I forgot who demoed on one of the Facebook conferences, and I thought that looks a lot better than what I'm doing right now. So I built a little prototype just kind of wrapping Stripe and GitHub it was really powerful. The two things that I struck me was, one, like how easy it was to specify the data I wanted. Then the other one was a kind of tooling. At the time, it was really easy to build like this waterfall visualizer to see what parts were actually slow in a network. So I didn't actually end up switching my product over to it. That would been a massive reengineering organization at the time. But it was very, very like powerful. It got very excited about it.

[00:03:50] JM: Describe how people are using GraphQL today.

[00:03:55] SG: I think the movement is largely driven from frontend engineers because it gives kind of product developers a much better way of specifying the data that they want if they're to build some specific kind of UI, whether that's mobile app or frontend app. Largely, the movement for it has kind of been pushed from the frontend backwards, whereas the backend people are just kind of now getting into it, I would say at a larger scale. If you're not inside of the JavaScript or Node world, probably GraphQL is at the top of your mind, whereas you probably started hearing about it from the people who are actually building user-facing products.

[00:04:30] JM: And do you think that's going to be the same forever, or do you think there is some functionality or some use case that backend people are going to care about in the limit?

[00:04:44] SG: I mean, it could go either way. I think for us we've seen it can work really well in the backend, but in the backend you're competing with solutions that have like really good properties. So things like Thrift or proto buffs or whatnot where you're using like server, server communication. This can be really, really optimized, plus you have teams that are really experienced with these toolsets. So it's a very rich kind of well grown out ecosystem.

So GraphQL is kind of – It founded a niche in the frontend where it could grow really strong, but it's going have a harder time as it works its way backwards. That said, the ability to be an engineer either on the backend or the frontend and say, "this is the data I want, and I want you to figure out the most optimized way of getting all of it to me as quickly as possible," is a problem that you face on either side.

Actually, GraphQL can work really well there. Then one other area that where we've seen it started to just to emerge is kind of in asynchronous APIs, in particular kind of webhook-based APIs, whereas previously you'd have something like, say, Stripe, where a payment is initiated and you'll get a webhook and it has a payload. The thing is whenever that payload comes in, you're typically saying, "All right. well, this is great, but I also need to pull off some data that's related to this." You'll immediately go into your database. Maybe you'll go into Salesforce and pull out some information about this customer who's just made some purchase.

Really, you're trying to express your tree of data, right? The whole data dependency there. What you want to do is say, "Well, I want a subscription and GraphQL, but I want to deliver it over

WebSockets. I want on deliver it over webhooks. When this event happens, I want these three fields and then go ahead and descend into this data in all three fields and just deliver this entirely to my endpoint so I don't have to go in and do any of that messy work myself." You've done a much nicer job of kind of encapsulating what are the data dependencies? What is your intense whenever this action actually occurs? I think that there is some unique properties of GraphQL that make it probably more amenable to some backend stuff, but there is a much stiffer competition on the backend.

[00:06:42] JM: As far as the backend, what do you think about the data migration use case? The idea that you could potentially use GraphQL as a descriptive layer to do ETL. Maybe you've got a couple really big databases on your backend with tons and tons of fields and you want to do some join between a couple of those databases and pull all the data into a new database or a data warehouse.

[00:07:08] SG: I think, long-long and maybe even medium-term, it has a really good shot at doing that because it is really descriptive in its ability of modeling out relationships between data. One of the challenges there is going to be that the most common use case is to – Most common serialization format is JSON, and if you're pulling out lots and lots of data from these databases and trying to join across them, this can end up being a really intense process especially at a large-scale. It's certainly doable with things like maybe a Pulse federation and some stuff that the rest of the community has been working on.

There is this idea of how do we do two things? One is officially model the data between all these different data sources and be able to efficiently join on them to avoid the M+1 problem and whatnot. Td then also there is the kind of more transport-agnostic approach, which is maybe we want to represent this in MessagePack or something like that whenever we're doing these kind of bulk automated migration jobs that require lots and lots of data.

[00:08:06] JM: As far as how people are actually using GraphQL today, as you said, it's mostly people on the frontend. What are the most common frictions in the developer experience of GraphQL?

[00:08:20] SG: It's hard to say. I mean, there are a number of them. I would say that one that I feel is somewhat surprising for people who are in the GraphQL community is that it's almost, by definition, by the time you're excited about GraphQL, you've lost the ability to explain it to other people. There's all this really cool tooling. One of my favorite is Graphical, which is like a little open source IDE. You can point it at any GraphQL server and you get this nice experience.

It has a ton of potential, and we've built out a bunch of open source tooling around it. But initially it's just kind of this blank code mirror canvas and it gives you auto completes, but it doesn't tell you how the syntax works or anything like that. You would find people that were into GraphQL demoing it and just showing like, "Look how cool this thing is," and start typing it out and auto completing, and they would give it to someone who didn't know that to who've just maybe seen this and they didn't know where to start. It's an entirely blank canvas.

We've seen a lot of the tooling has kind of been built for the people who are right on the other side of that valley. They've gone through. They're bought-in They're sold. They like this thing, and now the tooling will work well for them. But the onboarding experience hasn't been very strong, I would say, and it's largely been driven by I think some workshops. So the Noon Highway folks do a great, very empathetic workshop on GraphQL. But I think that probably onboarding is one of the largest ones at scale for GraphQL.

[00:09:41] JM: So GraphQL, for people who are getting used to it, has this component of the infrastructure called a GraphQL server. I am a frontend developer. I issue my query to the backend. It says I want the users. I want the number of comments from that user on this particular photo, and that query gets issued from – That single query gets issued from the frontend, and the GraphQL server can federate that query out to any number of data sources that need to aggregate the information for that query. So maybe you grab the data from the photos database, grab the data from the users database, you grab the data from the comments database. You pull it into one response and you send it back to the user to the frontend developer. So the developer experience on the frontend is I've just issued one query. I've gotten back data from one place. But in actuality, there's this server on the backend that's doing the federation of the query.

For the developer who wants to use GraphQL, are they typically standing up their own GraphQL server? Because I know that it can be kind of complex to set up that GraphQL server on the backend, connected to all your data sources, wired up with the right schema, etc. etc. Are people managing their own GraphQL servers or are they using some GraphQL as a service thing?

[00:11:07] SG: So what we've seen largely is – Like I said earlier, that GraphQL is kind of being driven from the frontend people, and largely this comes out of larger corporations because teams don't talk very well. They're kind of using this as a hack around Conway's law where the back and team has said, "Here's your API. This is the API you'd get," even if it doesn't actually match what you're trying to build on the frontend. It might be a very chatty API. It might be very under documented, that kind of thing.

You find that there's this BFF, the backend for the frontend team. This kind of pops up, where it's – I don't know how sustainable it is, but it's kind of a group of like a crack team of ninjas on the frontend who say, "What we're going to do is build a hyper customized GraphQL server for our product, and it's the one that's going to talk to all these other backend teams that don't want to give us what we want and it's going to take care of optimizing everything." Because we'll do this, that will entirely decouple the frontend team. So they're going to be running and happy and they can specify exactly what they want in the API and then we'll go ahead and add that on the BFF.

It's kind of frontend teams who have largely said enough is enough and started to stand up and maintain their own GraphQL servers. But your point, it is much more complex, and I think this is one of the challenges in a young ecosystem, is that the tooling isn't all the way grown out. Best practices aren't always specified. So it does take more work to actually think this way through.

The other thing that's maybe a little bit more challenging is GraphQL has this strongly typed schema that's introspectable, which is what makes really for the best tooling that you can have out there. The challenges, it requires you to think ahead, right? What are the fields that you actually want to have in here? How are they related to one another?

API modeling becomes more of a kind of front and center concern, which over the medium to long-term it's going to be a great thing. But when you're first getting started, you're kind of

desperate for success. You're just trying to get anything to work. In that scenario, like a Rails app or an express app with just an endpoint and a grab bag of JSON is really easy to work with. There is definitely this initial curve of both building and maintaining a GraphQL server can be more complex than the alternative.

[00:13:16] JM: It does seem like we're getting to the point where the frontend developer wants GraphQL from day one despite the fact that they only have one database. They just want all their queries to be phrased in GraphQL format, which has made me curious why there's not like a database service, like a backend as a service that is just – We're like a GraphQL as a database.

[00:13:49] SG: Yeah. There've actually been a few of these. Back in 2015 there was the first one, which was Reindex. It was a GraphQL backend as a service. Kind of a lot of people saw Firebase in the success there. Firebase is a great service, but they wanted to have kind of GraphQL and all the nice tooling and whatnot. So they said, "Well, we'll do just that, but with GraphQL."

First it was Reindex, and then Scaffold, and then Graphcool. Scaffold has since been rolled up into AWS AppSync, which is AWSs version of GraphQl backend as a service, which is really a GraphQL frontend on top of DynamoDB as well as kind of lambda and whatnot. So it is a very extensible architecture. Graphcool turned into Prisma and they've since had several different products out there.

Right now I think the two that are the most exciting – Because, for me, I think databases are one of the most challenging bits of software. The ones that you maybe want to take the least amount of risk on for most companies. It just so happens that we have this amazing database Postgres that is really solid that has been continued to be built out. Its open source. It's like a wonder of the world. It doesn't make sense to exist.

So there are two projects, Postgraphile and Hasura, both of which are phenomenal bits of engineering. I'll go into Hasura because I'm more familiar with them, but they're both open source, and you stand it up and you point it at your Postgres database and it will introspect the schema and actually give you a full GraphQL frontend, full GraphQL server with all of the

relationships, all of the joins and everything, all the tooling. They implement permissions and at a really robust way, very much like Firebase.

Basically, it turns any Postgres database into a real-time firebase with GraphQL including permissions and everything and it's open source. It's another one of those ones where like you're taking one good thing and adding another good thing and suddenly like it's amazing. You're like, "How is this free? This is crazy."

[00:15:45] JM: Before GraphQL, the main ways that companies expose their external APIs were through JSON or XML. I remember in college, I played with like the Yelp API a lot, and it cool. You'd query the Yelp API. It gives you back information about businesses. There were some other early open APIs. As time has gone on, the API economy has really developed Salesforce, which was originally just thought of as kind of SaaS product, or Zendesk, just thought of mostly as a SaaS product. People are increasingly using these as API backends, basically, what were you integrate your own software with them more intently. Then, of course, more recently, you have Twilio and Stripe that are full-fledged API as a service companies that have been extremely successful and developers have gotten a better and better experience, but there are more and more APIs to deal with.

Coming more recently, some of these companies, these API companies, have exposed external GraphQL API. So if I want access GitHub data, I believe that GitHub has their own GraphQL API. Instead of standing up my own GraphQL server, I can actually just issue GraphQL requests straight to GitHub. Why have company started exposing external GraphQL APIs?

[00:17:14] SG: I think there are two points here. One is kind of going back to that tooling and that experience, right? For frontend developers and actually anyone who is consuming an API, the first thing you're going to do is basically Google for docs on the API. You go to the docs, they're all unique. They're all Snowflake. I have to figure out how have you arranged your API. Which of these endpoints has the data that I want? Where in the structure is that data?

I then have to translate that doc into some sort of maybe a curl call, like just get started. I actually hit it. I'll figure out auth. Once I've actually successfully made a call and I've gotten some data back from your API, I have to figure out why it doesn't actually look like what you said

it looked like in the docs, and that I have to reconcile that. Then, finally, I have to get into usable context, which is like my app. The whole reason I'm reading your docs is to pull down your data to put it probably inside of my app or my migration script or whatever.

Because API design is so open and there are so many different ways to do it, it's basically opaque to computers. Things like open API and Swagger which attempt to add a computer readable layer to an API so they can give you some tooling. But generally speaking, there's a lot that's not captured in that, and the other big problem with that is that it's kind of best efforts, right?

We have had people who've approached us saying that they want to have a GraphQL API and they have open API specs that they're very proud of, and we've kind of just learned to say, "Well, actually, don't even bother with them, because we need to the ground truth. Not what we think the truth is, which is typically what open API specs kind of represent.

The GraphQL API gives these developers, these external consumers, kind of a guarantee about what things are actually going to look like and they get tools that can be leveraged everywhere, because every GraphQL server offers these basic introspection abilities, whereas the tooling for kind of REST APIs was bespoke, right? You had Apogee, and they would have like a nice little API playground, but that only worked if you had Apogee as your API Gateway, and it didn't work if you had anything else.

If I'm trying to build both my company and my API and a set of API tooling that is going to be pleasant for the developers who are consuming it, that's a whole lot of work, whereas with GraphQL, I can basically just rely on the community tooling that's getting better and better all the time and a list of people who know how to use it, and I can just focus on my API design.

The other one is it allows for some really, really good API migration. By default, the culture in GraphQL is to not make any breaking changes. It is a big deal. The thing is, that's easier because you actually know what will be a breaking change. You can download the schema for any GraphQl server. For us, for example, every time we push, the build will run and CI, and it will pull down the production schema and it will pull down the schema that we're about to push and it will diff them and will tell us, "If there are any breaking changes, and if so, it will break the

build," unless we have some specific commit message that we've actually communicated to the world saying, "Hey, this field is going away."

In fact, we usually don't even communicate it to the world at this point, because we're able to track at the field level which clients have access which fields. We know exactly who we might be breaking. So we'll actually just contact these people individually and help migrate them over to a newer version. That's an experience that is really, really difficult to get with kind of this more opaque grab bag of APIs. So what you end up with is a team of engineers who are deeply sad because they have this messy API that they don't know what they could change. They don't know what they might break. Anytime they do try to upgrade it, they actually end up breaking some people and they get upset and they get punished. So they just kind of want to leave it alone and no one wants to touch it, especially if that team has churned. If the original team that built it isn't there and you've kind of inherited this thing, it's a really challenging prospect to figure out how to improve it without breaking people.

[SPONSOR MESSAGE]

[00:21:13] JM: Gauge and Taico are open source testing tools by ThoughtWorks to reliably test modern web applications. Gauge is a test automation tool that makes it simple and easy to express tests in the language of your users. Gauge support specifications and markdown, and these reusable specifications simplify code, which makes re-factoring easier, and less code means less time spent maintaining that code.

Taico is a node library to automate the browser. It creates highly readable and maintainable JavaScript tests. Taico has a simple API. It has smart selectors and implicit weights that all work together to make browser automation reliable. Together, Gauge and Taico reduce the pain and increase the reliability of test automation.

Gauge and Taico are free to use. You can head to gauge.org to know more. That's G-A-U-G-E.O-R-G to learn about Gauge and Taico, the open source test automation tools from ThoughtWorks.

[INTERVIEW CONTINUED]

[00:22:29] JM: So what you are working on is related to the fact that in a modern company, you have a wide variety of third-party APIs. A mature company has probably Stripe, Twilio, Intercom, Salesforce. These are not databases explicitly to the user, but they do hold your data. So you do want to query them for data. Tell me about the patterns of developer usage where I am accessing multiple different data sources that are API data source. So if I want access Stripe plus Salesforce at the same time, what's an example when I might want to access both of those API services simultaneously?

[00:23:26] SG: A really common one might be, in the developer world, GitHub. GitHub is kind of like a – It's not always the central spoke, but it's almost always like peripheral. It's almost always one hop away. So we have people who we offer web hook subscriptions for the npm API, for example. Whenever someone publishes a package, you can get notified and you say, "I want you to deliver this payload to this URL, to this web hook, but in the payload, I want these three fields and also go ahead and drop down into the author and give me the author's name. Also, if this is a test to a GitHub repository, go over into GitHub and get the list of all the open issues and then deliver that as one payload to me on every npm package that's published."

Similarly, we have one where we have clients who use Salesforce's cases, and like having – I have done four or five Salesforce integrations so far, and it's a challenging API. It's like a family of APIs. As a developer, like you just generally don't want to go anywhere near there. In fact, Salesforce is a very powerful product, but developers don't even want to go anywhere in the product. For us, whenever we have to login to do some development, we actually have notes that we keep about how to get where inside of that to set the thing that we're trying to do, because we're not Salesforce UI experts. We just work with their API.

So we have a client who has recognized this and said, "Well, our salespeople want to create cases and they want to live in Salesforce, and our developers don't want to go anywhere near there. They live in GitHub. So what happens is we want someone to be able to create a case in Salesforce and go ahead and synchronize that over to GitHub very easily, and any comments on that issue should also be propagated as case comments inside of Salesforce while also not triggering any user limits and whatnots." There's like a pretty specific, pretty in-depth case.

[00:25:08] JM: Sorry. Not to interrupt you, but is this like my salesperson is in Salesforce. They're talking to a customer and trying to win the customer over and the customer saying, "Oh, you're going to need to build a new feature for us to actually pay you money again." Therefore, the salesperson wants to open up a case in GitHub and say, "Hey, implement this, or else I'm not going to be able to make my quota."

[00:25:34] SG: Very much like that, except that they do not want to go into GitHub. They know Salesforce, and so they want to stay in Salesforce. So they open up a case in Salesforce, and in developer vernacular, that's an issue. Someone gets assigned to that issue on GitHub and that's gets propagated over into Salesforce and just keeps – It's the same data. It's just kind of two different views on top of that, right? One that makes the developer happy. One that makes the salesperson happy.

Now, the teams are kind of independent. They can move quickly and life is good for everyone, I suppose. I still feel a little pain for the Salespeople who are in Salesforce, but I suppose they like it. But yeah, this is the idea, where they've kind of linked these two things and saying, "This Salesforce case actually has these GitHub issues as – Or GitHub comments and this issue is associated with this Salesforce case." They've kind of stitched together these different resources to keep them in sync with a little bit of additional code on top of that.

[00:26:26] JM: Okay. If I am that salesperson and I want to open an issue in GitHub, can you shed some more light on that? Am I using some internal tool at my company?

[00:26:40] SG: No. No. No. You just go into Salesforce. You don't even know that GitHub exists. You just go into Salesforce and you say, "All right. I'm going to open a new case on this opportunity over here, and we need to fix these three things and do this in order to close this deal by the end of the quarter. I want to be notified of like what the status is and who's assigned and that kind of thing." I go away and the next time I come and view it, I might see that so-and-so has been assigned to it and that there have been three comments. I just see this inside of Salesforce just like as though someone had been using Salesforce and adding comments.

On the GitHub side of things, I just see it as though the salesperson had come over to GitHub and open up an issue and we're talking inside of the comments, but I don't actually go over –

There is an internal tool that just does the syncing between the two of them using OneGraph, but it's just pulling it from this graph and pushing to that part.

[00:27:28] JM: Got it. This illustrates what you're building with OneGraph. Basically, the idea is you can essentially sync the data between the two. My guess as to what would be going on is there's some server in the OneGraph world that is continually listening to changes on the Salesforce side. It's continually listing the changes on the GitHub side and it can issue basically updates to one side or the other that there needs to be some data syncing to happen.

[00:27:58] SG: Exactly. We can kind of Eventify underlying APIs that are not even event-oriented, or in the case where they are, we know how to set them up very efficiently. For GitHub or Stripe, they have a great webhook system. For Salesforce, this installs – Or it involves creating Apex class and installing triggers on the objects and whatnot, but we'd all do that behind the scene. All you know is that you're in Graphical and you've issued a subscription and you said ping me here whenever this changes and give me this data, and you don't have to know any of those.

The thing is we don't try to normalize the data model underneath them. We don't say that there is one user and it's the same user on Stripe and Zendesk and Salesforce, because then you end up with kind of the least common denominator problem where you can only represent the fields that are common to all of them.

What we do normalize is access to that data where you can just say, "Well, inside of Salesforce, here are lists of events, and when one of these events triggers, I want this data, and if this data is related to GitHub, I want to pull into GitHub and pull out this data and just send me a notification whenever this happens and do it however efficiently you can."

In cases where an API might not be inherently event-oriented, we can do pulling. For example, we have RSS through GraphQL, because I wanted to have an RSS reader in the command line. So I built a little – A curl utility that hits OneGraph and gets me a JSON feed of RSS, which is great. It's my podcast player in the terminal. It's a massively popular product. I'm sure everyone wants that. But –

[00:29:26] JM: You're just like my little brother. My little brother is always building these kinds of things. I'm like, "Don't build a podcast player in your terminal. Come on. Come on. Change your life. Change your life. Don't build an API."

[00:29:42] SG: I'm a recovering engineer. Yes.

Yeah. RSS, inherently, it's a thing you're supposed to pull, and people have set up kind of services around that to Eventify and whatnot and we're just kind of taking that same idea to say, "I have the subscription to this podcast over here. These are the fields I want. I only want the title of the podcast, a list of the items, and for each item I want to know the podcast link in the description. That's fine. If that ever changes, just hit me over here at this URL with exactly this JSON payload and I'll know exactly what I'm getting."

With OneGraph, we kind of push all of that down into the system. So you just have normalized access to it and you just say – You don't care how it's implemented underneath the hood. You just know that it's going to be executed as efficiently as possible.

[00:30:23] JM: Okay. Now that we've given an example, and that's a very concrete example, this syncing of data between Salesforce and GitHub. What is OneGraph?

[00:30:33] SG: Sure. OneGraph is – The pitch is very simple. It's a single API end points through which you can get access to all of the most important SaaS APIs. You can hit it and pull down customer from Stripe. You can get issues and tickets from Zendesk or nmp GitHub, whatever it might be. We bring together both existing GraphQL APIs, like GitHub, which GitHub has one of the best designed GraphQL APIs. If you're looking for how to design an API, GitHub is a great example of how to do it.

We also overlay GraphQL APIs on top of non-GraphQL APIs. Salesforce, for example, we're talking about earlier, has a REST API. They also have a bulk API and a batch API, and luckily most APIs now speak JSON. But there was a period in time where some Salesforce APIs only spoke XML, and Salesforce is going to limit you to 10,000 API requests per day. If you want to double that, it's $25,000 per year. So it's a big deal.

What's going to happen is if you are like a normal human being, a normal engineer, you're going to go read their documents, and hopefully within a few hours you're set up and able to actually make an API call, because going through their auth processes is intense. But once you have done that, you're probably going to go and read the REST API and say, "Oh, this is what I would use."

The challenge there is that it's a synchronous API. It is REST designed and its JSON, but you will very quickly – After you've built this prototype and you've kind of deployed it as an internal tool into your organization, you'll find that you will hits that 10,000 API requests per day very, very quickly. What Salesforce is going to want to do is actually not pay them more money. They want you to use the bulk API, or the batch API. But the batch API and the bulk API, depending on which one you use, is not REST- oriented. It uses their custom dialect of SQL, which is called SOQL, the Salesforce object query language. It can be asynchronous depending on which one you're using and it can potentially turn to XML. These days, it's better.

But if you imagine, if you've built this internal tool and you're like, "All right. That was a trial, but I'm done. I'll never touch this again," and then you deploy it and it hits that limits and you go and you look at this other API, it's radically different, right? Almost none of your code is going to port over to that because the data structures are all different. The endpoints are different. It's no longer REST it's SQL. You have to write like custom queries for this.

For OneGraph we can just say, "Well, we know what data you want. You've specified it in this GraphQL format. So depending on your latency requirements, we'll just compile this into and hit the cheapest, fastest endpoint for you." That's one thing that kind of having this intermediate layer, this kind of intelligence layer, is able to do for you, where if you've told us what you want, you don't care how it's actually getting it. You just say, "Here are my priorities. Lower the cost."

Another one that I'm really excited about that I think really should go away, like there are three big problems that bother me that shouldn't exist. One is rate limiting. The other is auth. Rate limiting is like I think a very, very difficult problem, because it's so nonlocal. You're making an API call and you get the result back and you just check if there's an error. If the error is a rate limit for 12 or whatever, you need to back off.

Now you have like – Depending on what language you're in, this is kind of maybe a callback. It's like a nonlocal reasoning. You kind of have to figure out how you're going to do that. But you not only have the back off right here, you might have other calls in the same process that are hitting it that also have to be aware that they need to back off. You also have other workers, other processes, other applications that might be hitting this same API.

You actually need global reasoning to achieve really good response to rate limiting. Usually as a developer, like you don't actually care about this. What you're saying is I want to make this call, and whenever the data is ready, just give it to me. I don't want to actually be pulling you or whatever it might be.

So what we can do is actually say, for example, with Gmail, Gmail also limits your API calls. We have people who writes tools that pull down data from Gmail and that can potentially lock the user out of Gmail, because you've actually exceeded your API limits or whatever.

[00:34:42] JM: Oops.

[00:34:44] SG: Yeah. Which is not always the most pleasant thing. So what you want us to say, "Hey, I have this big data. I know it's going take a while. I'm not latency-sensitive. What I want you to do is to automatically paginate through all of it. Go get all of the data, but never let my remaining API calls for the hour drop below 5,000. That's the priority. Then whenever you're done, just go put all of that somewhere in S3 that I can go and hit and I can hit it as hard as I want." Now I can kind of detach the local reasoning and I can just like hit it as hard as I want.

There are a series of these kinds of intermediate things that you can build that mean as an engineer I can just focus on the problems that I care about. If I am latency-sensitive, then I can specify that and I'll deal with it. But otherwise, the point is I just want this data and I can express it even if it's across all these different services including my own database, and whenever it's ready, I'll go ahead and operate on it.

[00:35:35] JM: That's really cool. So just papering over a lot of details to give the listener a chance to catch up if they feel confused, if I may get set up with OneGraph, that means that you're going to be spinning up some infrastructure on the backend that's going to essentially

give me as the developer the ability to connect my Salesforce and authenticate it and whatever. Connect my GitHub and authenticate it and whatever. Do whatever indexing is necessary on these Salesforce and GitHub things. I could do the same with Stripe or Twilio maybe or Intercom, whatever. Then I can just query OneGraph, and OneGraph is going to do the necessary GraphQL translations into queries or subscriptions or whatever to these APIs and you're going to be doing a lot of painful papering over of the rate limiting and whatever Salesforce API – Synchronicity. Synchronicity, those kinds of issues, to basically give the developer a magical experience.

[00:36:43] SG: Yeah. I would actually say kind of OneGraph has couple of different phases. The first one that we found, our initial hypothesis was that people would want to join across APIs. That would be like big think, because that's what I was doing. What we found is people usually come to us as like the best experience for getting a specific API and they only later ended up pulling in cross-API data.

For example, we have quite a few Salesforce developers now who use us because we've built, for example, an almost file system explorer where it says here are a list of all the services. So you just kind of open up Salesforce and it says here are all the subfolders. You say, "I want accounts, and for each account I want these three fields," and you hit play and it'll ask you, "Oh, you need to login," and it'll go through auth flow. You hit play again and you have your data." Then we'll actually analyze the AST of that query that you've just built up and we'll say, "Hey, we can actually generate you a React component if you'd like, or a curl script if you'd like, or a reason," so that you can just kind of go through, you specify the data you want and you see the results. You're like, "Yup, that's what I want in my application," and then you generate the code. You copy and paste it and has all of the best practices in there.

So they've gone from kind of exploring the API to actually having it inside of their app in about five minutes versus several days usually, and these are like people who are actually really experienced with Salesforce. That experience is significantly better and they're much more efficient than they would be otherwise.

We were kind of surprised that actually just by building lots of open source, really good tooling, around the developer experience for onboarding an API, you can actually get a lot of people

who are excited about that. The next step is maybe auth, because auth is another one that I think should kind of go away in terms of it's a really difficult and painful problem. Everyone is kind of unique.

OAuth had done a really great job of normalizing it to some degree, but I really just want as a developer to copy and paste some code snippet that you have in your docs, paste it in and actually have auth already working for me if possible. We've kind of implemented that. Then the last step is once you've like understand this value proposition tools like Hasura become really, really valuable.

As I mentioned before, you point that to any Postgres database, you get a real-time GraphQL API. But what they actually allow you to do as well is import any other GraphQL API's. So now you have this other API right alongside your database API. So with OneGraph, that means you can hit Hasura and it will pull in your database API and also your Salesforce APIs For me, the most exciting one as they have this feature called remote joins now where you have a user table. It has ID, first name, last name, email, and it has this field, Stripe customer ID. The only reason you have Stripe customer ID, the only reason you ever have it is to get to the Stripe customer.

The problem is, as an engineer, you query into your database, you pull out this user, you pluck off the Stripe customer ID and now you have to go and call the Stripe API and join them, right? You're manually traversing that graph. There is a connection there but it only exists inside of your head. What they allow you to do is say, "All right, well you've pulled in this remote schema. Do you want to add a field to your user?" It will add a Stripe customer field and you say, "Yes, the Stripe customer field is in OneGraph, Stripe customer," and a customer is parameterized by its ID. You just say, "This ID just comes from the same row, Stripe customer ID." Within about 10 seconds, you hit save, you now have a Stripe customer field on your user. So you query into your database, you get the fields you want, email, first name, and you're a Stripe customer and then you descend into charges and you get a list of all the recent charges.

So you're able to actually traverse from your database into these external APIs in an incredibly quick way. That's that is so much terrible work that just goes away, and that's only possible because of the introspectable nature of GraphQL. I actually think that that's probably one of the

larger wins in terms of like cross-service, that services are really your database and this other service. So being able to join at that level. Then you have lots of those where your database is kind of the central spoke and then they have kind of peripheral nodes around them. So a user has tickets inside of Zendesk and opportunities inside of Salesforce and charges inside of Stripe and packages they publish in npm and whatnot. So being able to join like that is a really big deal, but typically only for our larger users.

[00:41:05] JM: So your life is just unending pain, basically, because all you are doing, your day-in and day-out experience is like how do I overcome rate limiting in Salesforce?

[00:41:17] SG: Yeah. I mean, those interesting. Even just setting up subscriptions required a very deep dive into like Apex classes and database – Or salesforce triggers and whatnot. But once you've done that, once you've chewed on that glass, suddenly everyone else is like, "Yeah, the choice between these two experiences becomes obvious. This is the one I want."

We do joke every now and then, it might be nice to have like API design horror story hour where like we've kind of seen things, we're just like wondering how – It's very interesting, because you can see almost Conway's law just seeping through the API where we've seen one of our partners came on early on, they had open API specs and we implemented them and ended up actually just ripping it out. This is one of early lessons that I – The ground truth.

With GraphQL, if you have a schema and you say that this field is a string and it's not knowable, I mean, you are never allowed to return anything other than a string right there. If you do, you have to actually just failed the whole thing. You can't recover from that, because the client should never see malformed data. That's your responsibility as a GraphQL server now. So we had to rip all that up because it turned out they were just wrong.

In addition to that, we saw – They had a date time in their schema and the thing is they specify it's millisecond since Epoque or something, and we found seconds since Epoque, milliseconds since Epoque and RFC 3339, like three different forms of date times. Then most of the fields were camelCase. Yeah, camelCase, I guess, created at. Any date field had an at in a nice capitalized way except for some didn't have the at at all or they had created_at.

As a developer, like you've hit two of these endpoints, like, “All right, I'm starting to get a feel for this API.” Then you hit another endpoint and you go to pull out poor created at, but actually this time it's just created or it’s created_at. We just went through and what we do this we saw a set of tooling that just watches the API. So either as a client or as a server and it builds up actually a model based off of Novelty of all of the endpoints, all of the structure of the API and then spits out the ground truth. This is actually what we saw in reality including the graph relationships between these different APIs.

[00:43:45] JM: The idea is really cool. The idea that you have this singular backend that accomplishes joining across data sources that are painful to work with and it removes a lot of friction. Now the other kinds of tools that this brings to mind, you got Zapier, which makes it easier to have API requests to – Well, no. Actually, Zapier I guess is different. I guess Zapier makes it easier for non-developers to work with API systems and create like connectors and stuff. It kind of reminds me of RapidAPI. I don’t know if you’ve seen that, but that's like a singular – It kind of brings all of these APIs together into one service and you can have a singular authentication path that lowers the friction to the API development. These things seem great. What I wonder is how do you get people to adapt it? It's kind of a different paradigm. I think developers are generally used to going out to each of these different API services. They’re used to going out to Twilio. They’re used to going out to Stripe. They’re used to going out to GitHub. They’re not necessarily used to going out to Salesforce, but the ones who do have sophisticated big Salesforce integrations, the company is big, it's probably harder to introduce a tool like OneGraph. I'm sort of going in a lot of different directions here, but I'm kind of getting towards the business side of things just because I've seen a lot of developer tools companies.

[00:45:24] SG: Effectively go-to-market.

[00:45:25] JM: Go-to-market. Yeah. How do you get people to use this?

[00:45:29] SG: Yeah. It’s really tough. What I wanted to do – So I built the first version of this around January-February of 2015. That's when I saw this and I thought this is really exciting and I thought it was going to be at least five years before there was a market big enough to actually sustain companies around this.

[00:45:48] JM: You're talking about GraphQL abstractly.

[00:45:50] SG: There's just so much customer education required for this. Actually, on several different levels, there's GraphQL itself, but then there is also just the idea of your data is a graph rather than different silos. If you work at kind of older larger companies, say like a McDonald's or an Allstate or whatever in their IT, they're not used to thinking of it as a graph.

So what happens is there are actually all these problems that they have that they think are different problems and they try out to be the same one, which is being able to discover and describe data effectively. Td these problems are authentication, authorization, rate limiting building applications, whatever it might be. At the core, this is the problem for it.

If you approach them and you say, "Hey, I have this graph and it solves all of your problems." That is the same as saying it solves none of your problems, because they won't know where to start with this. So what you have to do is actually slice off very thin, very specific pain points and say, "We'll solve this one thing for you," and you're not going for mass, mass market. You're trying to find the ones that are the highest leverage, but still none of them are going to be huge. So you're going to take that and you're going to solve it for them and then you're going to grow them on the platform and say, "Well, actually, know how auth was really painful for you. That's because you have hard time getting access to your data." OneGraph had to implement auth for all of these services in order to get access to the API. So we already did auth. So you're using our auth. That's great. But we also have access the data if you want. You can just go through and actually start to maybe get your data from OneGraph and now you're hitting rate limits. You can actually push that down into us as well and you want this to be an asynchronous API that hit your service. We have basically just said, "Well, what are the use cases that we can find developers where they hurt a lot right now and just solve this one problem for them and then grow from there?"

As far as getting into the larger companies, our approach has been very bottoms-up. Neither my cofounder nor I are avid salespeople as evidenced by our lack of experience with Salesforce maybe. So we wanted to have kind of this bottoms-up movement. I used to work at Sauce Labs a long time ago which had also an open source bottoms-up approach, which was really painful

in the beginning, but it's a much more effective way of selling into organizations if the developers actually like you, if they're pushing for you.

What we've started to find is that we've had some large Salesforce related system integrators that have their own tooling that they're getting pulling from their customers, from their frontend developers saying, "Look, we have our own GraphQL API, but we really want to be using OneGraph for Stripe or for Salesforce, whatever it might be." That is the only way that you get those companies actually motivated to bring you in and sell alongside them. You can't really do that top-down. You can convince the CTO that, "Hey, this is a great idea," because everyone is going to be by default of inertia against it. They have things that are already working. They have their own solutions in mind. They don't want to learn about this new thing. But if their customers are telling them, "Hey, this is what we want. Why don't you have this?" Then that makes all those conversations much, much easier.

Our approach is been basically build this thing out, build lots of open source tooling around GraphQL will to make it as nice as possible and then just go and work with all these company. GitHub, for example, released a new version of Graphical that uses our Graphical Explorer open source extension that we've put out. AWS has done this. Shopify has done it. For us, it's just been about, "Hey, put this out there. Increase the GraphQL ecosystem as much as possible. Make sure that we're always kind of mentioned alongside any of these announcements as much as possible so people become aware of us." Then as they become aware of us, they start to use us. As they start to use us, they wonder why they can't use us inside of their org where they're actually working on this stuff.

We've also done a lot of call it hobbyists things. So we've built a tool called SpotDJ, where it's a pretty cool like ReasonML web RTC client-side only thing. It uses OneGraph to actually control your Spotify payer and get what you're listening to and then gives you a link that anyone else who comes to that link will be controlled by your Spotify player. You basically turn into a DJ, right? It's like a peer-to-peer Spotify DJ thing. We built that just as a proof of concept. It was like a Friday and we thought, "Well, we build a little fun app," and we found people who come in and sign up for that and they're looking to the Spotify API and then we get a message saying, "Oh! We notice you have Intercom here. How can we use Intercom?" Do we have to pay for that?" Because they're at work right now and they're supposed to be working on the Intercom API to

pull this in for our dashboard. But it's painful and they didn't want to really deal with that and now suddenly they're like, "Oh! I'm already getting data out of Spotify. Can I get it out of Intercom as well?" and they're kind of clicking around.

Gatsby, for example, has pushed us as a good data source for Gatsby, because Gatsby is this static site generator. It has these GraphQL sources and usually they're just individual sources. You have the Stripe source or GitHub or whatever or you can just have OneGraph and you have all of them available. You can just query into them. So they actually built a Gmail application with us, with Gatsby and OneGraph. It's a Gmail client. It's very nice, and that send us a bunch of people who were looking at the Gmail integration specifically, but then they tend to kind of get drawn over into the Salesforce's or QuickBooks or Crunchbase. These APIs that they're supposed be dealing with that are kind of famously challenging.

[SPONSOR MESSAGE]

[00:51:31] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:53:20] JM: Do you worry about a world where all these different companies set up their own GraphQL APIs and then there's perhaps no use for OneGraph?

[00:53:32] SG: I don't think so. I mean, there could be some worlds in which case we'll have to find different ecosystem to live in or different niche.

[00:53:39] JM: I guess even in that case, you're figuring out the join dynamics.

[00:53:43] SG: Exactly, none of them are super incentivized to join them together on their own. GitHub doesn't want to be responsible for Salesforce's API or the joins between them. I presented you my data. There should be stuff like you figure out how to join it. I think that that's probably the ecosystem that we inhabit long-term. But even on top of that, you have things like auth and rate limiting and tooling and whatnot that you want to have, for example, an internal tools team outside of your organization. Wants to have access to all these stuff and they don't want to be thinking about Saleforce's rate limits for service cloud or Zendesk or whatever. I think that having this API tooling available as like an internal tools platform or developer tooling. It's still probably okay even inside of a GraphQL first-world.

[00:54:31] JM: How do you avoid the rate limits of Salesforce? If you're making all these requests to the Salesforce API as OneGraph, wouldn't you eventually get rate limited?

[00:54:43] SG: Ultimately, everyone will get rate limited, but we're making the requests on your behalf. With your OAuth token – That's fine.

[00:54:51] JM: I see. I have to go into – Well, Salesforce gives me no OAuth token.

[00:54:56] SG: Yeah. All you have to do is click sign in and we actually go through the OAuth flow and provision everything for you.

[00:55:01] JM: Got it.

[00:55:02] SG: Then now we have a token that is going to be counted against your rate limits, and now – You still have rate limits. It's still possible to exceed them even with OneGraph, but we can do things like optimize it to hitch the bulk API and pull out 10,000 or 5,000 objects in one query rather than 5,000 queries. So we can give you a pretty huge leverage there.

There are cases where we do data virtualization for some of our customers where we basically pulled into our own database. It's almost like ETL or like a data lake, but for real-time applications. Typically, these things are used for like business intelligence and whatnot. One of the models that we use for these APIs that are very difficult. So QuickBooks, for example, is very challenging in lots of different respects.

What we do is where she pull it into database hosted on them, the customer as infrastructure, and we provision automatically all the webhooks upstream and then we give you a real-time API to that – Or access to that database. Now you can do queries that you couldn't do on the original API.

A good example of this was Stripe for the longest time. It's a payments company and they have a great API. You can list your invoices with their API. A thing you might want to do as a business owner is list your unpaid invoices, and you couldn't do that, right? You had to download all of them and then filter over them. I think they added that earlier or late last year.

But as an example where unless the API provider has thought about this specifically ahead of time, it may not actually be amenable to what you want to do. QuickBooks is a very similar API where it's limited in many, many regards. We pulled in and now we give you basically this full ability to slice and dice through this data with no rate limits. Then we expose mutations.

GraphQL has three different concepts. There are queries, which are the pure read-only operations where you're getting data out. You have subscriptions, which are I want to get data

out, but also if anything changes, let me know. Then the last one is mutations, which are the effects. This might be initiated payment, pause my Spotify player, go to the next song or revoke this invoice on QuickBooks. Now what we do is whenever we expose those, in turn we've instrumented in such a way where we know which data is actually going to be dirtied upstream. We'll run that mutation and then immediately re-synk that data into our local data virtualization database so you always have this real-time fast API that you can hit with no rate limits, or the rate limits are effectively the infrastructure that you allow for this database. Those are only the more extreme cases where we can't figure out how to actually make the backnd API amenable to this. But whenever we do something like that, it's very high-leverage for the development.

[00:57:50] JM: When you say data virtualization, what I'm hearing is like you query their Salesforce data and may be eagerly grab as much as you can and cache it whatever, Redis, or Mongo, or – I don't know, S3 or something on your side so that when the client queries again, you may be able to avoid another rate limiting deduction by just hitting your own local cache.

[00:58:22] SG: Exactly. What you want to avoid there though is the stale data problem where you have a cache of its, but upstream it might have changed and that was no longer consistent. In order to do data virtualization, you have to be able to detect changes upstream efficiently. This is where actually the same technology we used for subscriptions comes in handy, because we have to be able to easily set up in Slaseforce the Apex classes and triggers. QuickBooks, it's the webhooks, Stripe's webhooks. With npm, for example, it's –

[00:58:48] JM: That's like where – Sorry to interrupt you, but that's like where Salesforce or QuickBooks basically says, "Look, if there's a change, we're going to push it down to you."

[00:58:57] SG: Exactly. Often times, the change is the most opaque thing. It's like, "Hey, a thing changed and it's up to you to figure out what changed," but we're paper over all of that. Npm, for example, they have a pretty interesting model where they're running on Couchbase and you can actually just spin up your own Couchbase instance and connect to the change feed. There are lots of different ways to detect the changes across all these APIs. We push that down into the subscriptions and then we're also able to leverage that for data virtualization.

[00:59:28] JM: Tell me more about your backend.

[00:59:30] SG: Sure. The backend is primarily written in a language called ReasonML. Just a difference syntax on top of OCaml.

[00:59:40] JM: This is what Facebook uses internally for something?

[00:59:42] SG: Yeah. Flow is written in OCaml. A number of projects are written in either Reason or OCaml. Messenger.com is written – The web frontend is written in ReasonML, and it's a very nice language. It has lots of positives. Lots of negatives, like any language. Yeah, it's largely that. It's largely self-contained. It was built from day one, because our business model is actually largely to sell to larger companies. We sell them actually a on-premise version of OneGraph that kind of spins up on Kubernetes and it's largely self-contained, because we have larger companies that have 1200, 1500 – Like numbers of integrations that normal engineers would think are astronomical. Who has 1500 APIs that they're integrating with? It's surprising that even that many exists, right?

For them, actually being able to discover the data is really important. That's where all these tooling comes in, and then being able to migrate it and monitor it overtime is really important. We saw this upstream and so we've tried to keep it as compact and as self-contained as possible, whereas if you have lots and lots of different services that are running that basically translates into support costs, and we want this to be as much as possible set it and forget it.

[01:00:54] JM: The primary user is the enterprise that you're selling a self-hosted Kubernetes instance. But if I am random hacker, I don't obviously want to install Kubernetes.

[01:01:07] SG: Yeah. No. You just use the public one.

[01:01:09] JM: Tell me about the backend infrastructure. Is it using lambda? You're spinning up containers? What you do on the backend if I'm a developer and I want to start using OneGraph?

[01:01:21] SG: Oh. I see. So not how OneGraph is built.

[01:01:23] JM: Well, I guess – I mean.

[01:01:25] SG: Happy talk about either one.

[01:01:25] JM: My understanding, why aren't those one in the same? If I'm a developer and I spin up an instance of OneGraph on your – I mean, you have a hosted version, right?

[01:01:33] SG: Yeah. You just talk to us over HTTP. You just make HTTP calls to us. Whether that's request in node or fetch in the browser or HTTP Kits and Closure.

[01:01:42] JM: Or are you just running – You just have your own Kubernetes installation running on AWS I assume and –

[01:01:47] SG: Exactly. Yeah, something like that. We're on a Google Cloud and we don't use Kubernetes. We think about it from time to time switching over, but my cofounder was the second engineer at CircleCI. So he very quickly just set up this nice – A very Kubernetes-like infrastructure on top of Google Cloud or computer engine, and now we could switch over, but the difference is like relatively minor.

[01:02:10] JM: Sure.

[01:02:12] SG: But it's the equivalent.

[01:02:13] JM: Side note. He worked at CircleCI. You worked at Sauce Labs. You guys both came from continuous integration companies?

[01:02:20] SG: Yeah. Long, long time ago.

[01:02:22] JM: Do you share any beliefs about infrastructure that you learned at those CI companies?

[01:02:29] SG: Well, I think all the standard ones now, cattle not pets. It's important that you – Any service should be able to fall over at any given time. You should have easy scaling triggers and whatnots. Both of us saw some interesting stories. At Sauce Labs, there was a point at

which I was – So I had joined Sauce Labs as an engineer and I became friends with the CEO during the interview with him. He asked four questions about you, because everyone – Whenever they're trying to hire someone, wants to know three things. Can you do this job? Will you be excited by it, and can we stand you? Those were the three criteria for hiring. He asked four questions. Asked them. Puts it down and he says, "Do you have a question for me?" I said, "Yes. I looked you up, you have my dream house, and I will never be able to afford it working for you, and I don't understand – If I'm at point A and that's point Z, I don't understand the path to get there, because right now I'm not making anywhere near that much money." He laughed, and about 8 months later he put me into sales. He said, "You're good on the tech side, but you need to understand business." I went into the sales side, and I was trying to close a deal as a sales engineer. Handling the technical side of things.

[01:03:39] JM: This is Sauce Labs. You're trying to sell people continuous integration software.

[01:03:43] SG: Yeah, basically. But the bigger org go into, the more restrictions they have, the more criteria they have. We were on AWS at the time and it was very, very expensive and we were looking to switch over to our own infrastructure. That was going to be a long process and we had bought the servers and we had install parties and whatnot, but it wasn't ready.

The end of Q4 is approaching and I've taken a cab out in the rainy night in Mountain View to this place. It's like 8 o'clock, and there's actually a Selenium meet up happening in the Sauce Labs' office and we needed to have a dedicated IP, and the way we're set up we, couldn't do that on AWS. So we decided during that meet up with two engineer sitting in the back behind all the stacked up chairs to switch over the live production over from AWS to our own untested infrastructure. It's probably not a thing that we'd do now. This is a long time ago and those were some pretty superb engineers. Everything went amazingly and it literally probably saved the company in some respects. It was a huge deal to get this over in a couple of different ways. That was the result of having a very – The guy who set up had this philosophy that if you are spending your time in the data center, it's because you failed. Everything has to be remotely administrable. Everything has to be like cattle. It has to be easy to throw away. It has to be great for monitoring and whatnot, and it's s probably not true at their scale now. They're huge now. But at the time, that taught me a lot about how to actually approach any of these – Building any these kinds of systems.

[01:05:22] JM: Getting back to developer tooling, but also touching on that same subject, do you have a sense for – Well, maybe you could tell me from your experience as sales engineer, and this touched on what we said earlier, like what is required to build enough developer appreciation to actually successfully go to market with a developer tool? From your experience with Sauce and from your cofounder's experience with CircleCI, how do you actually successfully go to market with a developer tool?

[01:06:00] SG: It's definitely challenging. Developers – And it is challenging to some degree because developers have more and more purchasing power, whereas maybe a decade ago they actually had relatively little compared to the numbers that you need to do as a business. I think a mistake I see a lot of very technically-minded, very capable engineers, make when they kind of formulate a business idea is that because there are other developer tools that have made lots of money, it must be okay. As long as you can make developers more productive, then potentially you can make lots of money.

But there is a really big challenge in translating and like identifying the specific value prop, like why is it so important that you're making these developers productive in this way? What is it that you're removing? How does that translate up to KPIs, key performance indicators, for people at the top of the company? Because it's very common a CTO at bigger org has no idea about the technology that's actually running inside of their org. All they see is kind of the outward metrics, and that's the thing that they're focused on the most.

So the fact that your thing compiles 2-1/2 times faster than this other thing and that saves three hour, developers a week, that's going to go through several layers of abstraction and it's just like a minor point variation at the very top level, which might be enough to actually make lots of money, but you actually have to be able to figure out what's the story. The story that you're telling to the developer is very different from the one you're talking to the engineering manager, to the VP of engineering, to the director, to the CTO. I think that the first step with the bottoms-up adaption is to definitely build something developers love and – I don't know. I really like magical experiences, right? Something that – Maybe a good example here is my first time through YC. Paul Graham was there. It was 2011, and he had this knack for he would say – He

would ask you something. It was like he knew nothing about your company and he would say, “Hey, does your company do X?” like, “No. Of course, it doesn’t do X. Maybe it should.”

At one point he – The company I was running at the time was cloud infrastructure. You could spin up any open source app very, very easily. It was easy to get like 13 different apps, and he asked, “Oh, do they all integrate with each other and to the users work across them? Totally different codebases. They’re all new open source projects. None of them know about each other at all?” and like, “Of course, you’re a developer.” You know what that would mean. That's insane.” It’s like, “Oh, maybe they should.” It’s like, “No. That’s impossible.”

Then you kind of think about it and you think, “Wow! That would be a really magical thing,” right? If you were able to spin up an open source project and it had access to all of your data and it could work with all the other apps and it had your users and if someone left the company, they were actually removed from all these different apps. Wow! That would be a really magical. How would you do that?” That's actually what got me started with going down the integration path.

I think that kind of – It’s so obviously impossible that if you can even achieve any semblance of it, it will feel magical. If you can even just get a little bit of it and someone will say, “That is so different from what I do today. I want that. How do I get that?” I think that's like a really good first step.

[01:09:13] JM: And it does take a really long time. If you look at Zapier, like today, that's kind of what they have in terms of the magic, and it took a pretty long time for that company to really I think gain broader acceptances like, “Okay. This company is going to be going places.”

[01:09:30] SG: They hustled so hard – They did so many non-developer related things. They worked so hard in the SEO and content and just getting in front of the people at the right place at the right time. As a developer, that can be very, very uncomfortable. Often times, building more on to your product is actually bad for it. It’s going to make it harder to understand. It’s going to make it harder to sell, harder to find your right customer. What you need to do is go and actually sell this thing. Go find whatever channel it is to get to customers, and that's – San Francisco and Silicon Valley is maybe a little bit unique where you have way more developers than you have people who are willing to go out sell and idea people. I think it's because it's just

a very uncomfortable thing. You feel like a master of your domain whenever you're developing. You feel like a novice whenever you are going out and doing this thing that you're not used to having to do. Also, you get rejected a lot.

[01:10:24] JM: Do you feel like you're catering to GraphQL developers or are you more targeting people who need integrations?

[01:10:33] SG: The hypothesis is that long-term there'll be the same thing. I think that 2020 was originally – I built the first prototype in 2015. I thought 2020 was about the right year to launch a GraphQL company. There would be enough awareness that could actually grow with them, and given the tooling that you could build, that was already kind of visible with GraphQL back in 2015. You could imagine all of the other things that you're going to be able to do and eventually the momentum of that would actually sweep over and anyone who wanted integration, this would be the way they would do it. But you could get into the market through the GraphQL ecosystem.

[01:11:09] JM: Beginning to wrap up, we've talked about some the other GraphQL companies. There's Apollo, Hasura, Prisma, Scaffold got acquired. Give me your perspective on the landscape of GraphQL businesses and any other gaps or opportunities in GraphQL tooling to the point of what you said, "Okay, if it's five years after GraphQL got started and today's the day to start a GraphQL company, what are the gaps in the tooling?"

[01:11:43] SG: I think the wider ecosystem is getting much better, because I think people saw originally the success of developer tools, Firebase and whatnot, maybe Meteor, and misunderstood what that meant for a business. So they started GraphQL businesses before there was a market. So that leads to people kind of raising money, often expectation that they're going to be the next Parse or the next Firebase. When that doesn't materialize that leads to a potentially kind of negative outward effects. You're desperately to claw any sort of success.

I think the early ecosystem had a lot of variance in terms of how things were going. There'd be some popup here, but then someone would fork a thing, rebrand it and try to call it theirs. That has settled down to some degree. I think the ecosystem is starting to get much better. I would say overall, the landscape is much better.

As far as opportunities, there's going to be a lot in terms of basically anything that exists for the REST world, right? So documentation, API monitoring. There's kind of always going to be an analogy over here. The question is how much of that is going to be owned by Apollo, which has had a great head start and has raised I think $53.2 million at this point, right? It's a massive, massive company. Well, massive by I think my standards. There are certainly larger ones.

I think on top of that, there's going to be some really exciting ones that people don't realize yet. One for me that I think is crazy is GUI builders. I think the way that GUI builders, tools that actually help you build UIs, operate right now is very, very limited and it's largely because of – I would say maybe the nature of JavaScript makes it very difficult to introspect. The computer can reason very little about JavaScript. So it can't really help you very much. The best it can do is maybe run some JavaScript and then tell you what the result is.

The problem is the JavaScript could do anything. It could make network calls. It could drop the database and the nasty emails of your boss. You don't know what it is. You have to be very, very careful whenever you're running it, and this leads to tooling that's very, I would say, cautious, maybe conservative. The other problem is the GUI builders are largely focused. They're just focused on this one problem right now. It's so difficult to do that. They haven't even thought much about the rest of the world.

I actually worked on GUI builders a little bit in 2014 as kind of a little hobby and I was experimenting with it and I found that there were lots of things you could do if you could inspect the language, but the problem was you couldn't really go beyond the GUI. Ultimately, the purpose of this tool is probably to talk to some server somewhere. But because you don't know anything about the server, that's where the tool ended. You had to drop out at that point and start coding. Coding is great, but it's relatively lower leverage if you can have a tool that will help you with it.

With introspectable servers, this means you can imagine like a GUI builder where you're building a title screen. You say, "Well, this title right here is actually coming from the server.user.address.street," and you don't even know that you're writing GraphQL. You don't care that it's GraphQL. You're just saying, "This is the data that's available on the server and this

is the data I'll actually want to specify right here." Behind-the-scenes, it's actually able to generate as hyper optimized GrpahQL called 4U. Not just a GraphQL call, but actually persistent that to the server and say, "Hey, server, I'm about to build this client. Here is the exact query I need. Store it for later," and the server will give you back an ID and now you're only going to send over this little hash of that query in the future. It's all hyper-optimized. There are many like powerful things that can happen there that you don't even know about. All you know is that you're just building a GUI and it's faster and you're happier than you were before.

I think mocking – Mocking is going to be insane. One thing we do right now is we don't store data, but as it passes through our pipes, we actually pass each field to a classifier and we just say, "Hey, what does this look like to you? Just some random bit of data. What does it look like?" The classifier will say, "Well, that looks like a number. Don't know anything about it. That looks like – I know that's a string and that looks like an email. That's a valid email. That looks like the first name. That looks like a full name," and then we basically just bucket the probabilities per field.

At some point, you're going to be able to actually have that GUI builder or any other service that's tying into it and say, "I actually don't even have an account yet. The developer team is still working on setting it up, but I actually can generate very realistic data through here, because I can generate first names, and addresses, and emails, because this all just pushed down into the system. So I can just experiment and actually build this thing."

Now once I've built this with the assumptions that I've had, I can break those. I can test those assumptions. I can say, "Actually, rather than Sean Grove as the name, show me the top thousand common English names. Now, show me Chinese. Show me right to left. In fact, fuzz my UI. Take every bit of data and fuzz it and let me see just the visual diff," and I'll see things that have moved around in ways that they shouldn't have because maybe double-barreled surnames are longer than I realized I think it actually be. Maybe in Spain that's going to be a problem, and I can actually fix that right now rather than waiting for a developer or a user who is actually nice enough to report this bug. Being able to actually hold this data of GraphQL is going to enable a bunch of different opportunities, I think.

[01:16:53] JM: Sean, thanks for coming on the show. It's been great talking.

[01:16:55] SG: Thank you.

[END OF INTERVIEW]

[01:17:04] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it's definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[END]