EPISODE 343

[INTRODUCTION]

[0:00:00.0] JM: Firebase is a backend as a service. The key efficiency of the backend as a service is that it enables developers to go from having a three tier architecture, which is a client server database, to a two-tier architecture, which is client and backend as a service. In this episode we discuss that transition. The team who started Firebase built it as a pivot.

They had started a social network and then they realized that there wasn't a good backend for the chat tools that they want to build, so the started a chat as a service tool for people who want to include chat in their applications. That led them to the fundamental realization that chat is actually representative of a broader category of real time synchronization problems. Firebase was eventually acquired by Google. Google has taken under their wings and really grown a lot of products on top of it.

Doug Stevenson is my guest today. He's a senior developer advocate with Google and he's the host of Meet Firebase, which is a YouTube talk show about Firebase. It was a real pleasure to sit down for a conversation with him especially because I recently started using Firebase in my own application, Adforprize, as a backend for real-time chat. This was a great conversation. I hope you enjoy too. I'm going to do more shows about Firebase, because I really like this product.

[SPONSOR MESSAGE]

[0:01:35.4] JM: Spring is a season of growth and change. Have you been thinking you'd be happier at a new job? If you're dreaming about a new job and have been waiting for the right time to make a move, go to hire.com/sedaily today. Hired makes finding work enjoyable. Hired uses an algorithmic job-matching tool in combination with a talent advocate who will walk you through the process of finding a better job.

Maybe you want more flexible hours, or more money, or remote work. Maybe you work at Zillow, or Squarespace, or Postmates, or some of the other top technology companies that are

desperately looking for engineers on Hired. You and your skills are in high demand. You listen to a software engineering podcast in your spare time, so you're clearly passionate about technology. Check out hired.com/sedaily to get a special offer for Software Engineering Daily listeners. A $600 signing bonus from Hired when you find that great job that gives you the respect and the salary that you deserve as a talented engineer. I love Hired because it puts you in charge.

Go to hired.com/sedaily, and thanks to Hired for being a continued long-running sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:05.8] JM: Doug Stevenson is a senior developer advocate with Google and the host of Meet Firebase, a YouTube talk show about Firebase. Doug, welcome to Software Engineering Daily.

[0:03:15.1] DS: Hey, great to be here, Jeff. Thank you for having me on.

[0:03:17.6] JM: Firebase is a backend as a service the. The key efficiency of a backend as a service is that it enables developers to go from having a three tier architecture which is a client, a server, and a database to a two-tier architecture which is a client's and a backend as a service. You kind of get to eliminate the database in many applications. Explain why this is useful.

[0:03:44.3] DS: Yeah. It turns out if your application developer you probably just want to focus on the experience of your app. That's what users are using. That's where you want to pour most of your resources into. If you have to worry about maintaining a backend, scaling a backend, paying for a backend, it's just extra time, it's extra money. It's generally unwanted, I would say, unless you really like to do backend stuff. In which case more power too. You do that thing. What we found is that application developers just want the entire package under one roof so you can build that app, scale it, grow it, make money off of it without having all that management of a normal backend.

[0:04:20.6] JM: The team who started Firebase built it as a pivot in their business. They had started a social network and then they realize that there wasn't a good backend for chat tools and so they started a chat as a service tool for people who wanted to include chat in their applications and this led them to the fundamental realization that chat is actually a representative of a broader category of real-time synchronization problems. Describe some of these are real time problems that exist in everyday applications.

[0:04:54.2] DS: You hit the nail in the head. Chat is the big one. I think that that's the one where every website needs some sort of a chat interface for support or just for user interactivity and that's great. It turns out there's a bunch of other applications. I remember I met a bunch of developers at Google I/O 2016 and they were using it for everything under the sun. One very memorable project was if someone wanted to send control or click command controls to a robot, like it was basically an IOT kind of a thing. They said that Firebase gave him actually that communication channel from one computer to another. It's just actually kind of taxing to build that yourself, and so if you have Firebase or Firebase real-time database in the middle, you actually eliminate all of that sort of scaffolding that it takes to get one client to talk to another. That's very hobbyist thing.

If you look more mainstream, you have applications like distributing stock quotes. Stock quotes are very well known to be alive, like very real, very real-time. What some companies do is they will have a database of stock quotes and streaming stock quotes, but to get that to the client actually push that into Firebase and then Firebase ends up being the delivery mechanism to clients of the web, android iOS. It doesn't matter. It works on all platforms. Those are a couple off the top of my head that end up having an advantageous real-time aspect of them.

[0:06:08.1] JM: The one that always comes to mind for me for real-time is Uber, where you're standing on a curb after you've requested your Uber and you're looking on your phone you see the car coming your way but the car moves in weird stutters and starts and it spins around sometimes. You know that the map is never quite up-to-date. I think of that application experience has indicative of just how far we have to go to getting real- time to be real- time.

[0:06:41.8] DS: Yeah. Actually, I gave a talk about that in that exact same issue back in I/O 2016. The name of the talk was Recipes for App Development with Firebase, and I proposed

two different apps and one of them was a ridesharing app. It makes it a lot more possible to build a ridesharing app where you have a driver and a rider. The driver's coordinates are constantly being fed into a location in the database, for example, and you have the client constantly reading those out. You have this sort of — It's almost like a collaboration app. When you think about it, the driver and the rider are collaborating on getting the rider to where they're supposed to be going. Those collaborative kinds of apps where you need that sort of real-time update so you can effectively collaborate is very much a real case.

[0:07:19.5] JM: Firebase allows for this real-time update propagation within the database. Explain what a user would have to do if they wanted to build real-time from scratch. Let's say user on the typical three tier architecture where you've got a client front-end and some — Like a node backend or a Rails backend and then a database backend and let's say they're building some kind of chat or ridesharing app. What would they have to do to get this "real-time functionality"?

[0:07:19.5] DS: Yeah, that's tricky. You would need to write your own middleware. You need to write your own service and you need to write ostensibly around special client to maintain an open socket and chat over that open socket. It turns out the way that Firebase does this is through web sockets. The client open up web socket to the service and there is constant chatter going on across that. That client would have to indicate that it's interested in a particular piece of information and then the server would have to say, "Okay. I know this client is interested in XYZ," and then it would have to essentially match up with other clients who are writing into that same location.

Now, if you can imagine how this works, like keeping an open socket is not that big of a deal but matching the clients, like the writer with the reader, would be extremely taxing, I think. Firebase handles a lot of that, a lot of the connection management stuff that you would have to handle on your own.

Another thing that you have to manage if you are doing this yourself is how to keep that connection open. As we do on mobile devices, connections are less than optimal. You're constantly bouncing between cell towers and jumping between Wi-Fi and cell and it's is never what you want, but users still expect their applications to work. If you had one of these apps with

a constantly open connection it would have to do this connection management where it'd have to look and see if it's closed, then reconnect. If it's not reconnecting, do a retry, do an exponential back off, and that logic is not exactly fun to write either. It's better to just have a set of client tools that manages all that for you.

[0:09:14.6] JM: Can you go a little deeper into how that bidirectional communication works? You mentioned a web socket connection between Firebase and the client application. Talk a little more about how that works.

[0:09:31.4] DS: That actually I've not dived so deep into that. I'm actually not the right person to talk about. I understand that everything happens over the wire in a highly optimized JSON format is what it boils down to.

It's interesting you asked that. There are people who don't work at Firebase who know more about that than me because they and up reverse engineering it because they wanted to build their own client or this one didn't understand how it worked. Yeah, it's a highly optimized proprietary JSON-based protocol that ships data back and forth.

[0:09:59.1] JM: I see. Do you know if it's like a — Is there a poling sort of thing going on, or you just don't know much about the adapter that?

[0:10:06.5] DS: I don't think there's a polling, there may be a keep alive, but I think the thing that's of interest is that the server always knows what the client is interested in. The client may say, "I'm interested in /users /some userids /some other location. You could think of it as path within the database and you're listening to that path. The server knows. The client wants to listen to that path. The server basically understands when rights happen into that path to distribute that to the clients that are listening at that path.

[0:10:33.3] JM: Okay. Yeah, that makes sense. If I'm building a real time chat room, something like Slack, and the chat room has 1000 users and someone sends a message to that group, the 999 other people will get an update because their client has subscribed to that message on the database and the one user who sent the message, their message gets written to the database

and then the database, well, Firebase says, “Okay. We’ve got 999 other people listening. We’ve got to fan out that message to those other clients.”

[0:11:14.2] DS: Yeah. In that example the chat room with 1000 users and it, each one of those users would be subscribed to that location. The reason why you're in that room is to get the update. On the client side you would subscribe to whatever the location of the database where all of those messages would be going.

It's funny, you think of it from an end-user perspective as sending a message and receiving a message. What's actually happening on Firebase is you’re synchronizing the message. When user one of the thousand writes a message, they’re actually writing it kind of locally and then that local copy gets synchronized to the server and then the server says, “Okay. I want to synchronize that update, or the entire contents of that location then out to the other users.” It's not like direct message passing. It's more like constantly keeping everyone up-to-date on any changes. As a result of that, not every client will receive every single individual message separately. They might, but they might just get an update with five messages that that's how much has changed since they’ve been offline.

[0:12:13.5] JM: Firebase is good for these real-time crud updates for like message passing or sending messages in a chat room. I think it could be more of a challenge to do things like real-time analytics across large volumes of data. For example, let's say I a photo sharing app on top of Firebase and I want to constantly calculate the 10 most similar looking cat photos across the entire app. Is that something that I would need to build a separate backend for, or is that something that I could do with the Firebase core functionality?

[0:12:54.0] DS: At lunch at I/O 2016, you most definitely could not have done that with the offering of Firebase at that time. Since then we actually had, and I think this was at the end of March at Cloud Next, we released the latest feature of Firebase which is cloud functions for Firebase. What that lets you do is write and deploy code to Google servers that respond to events within your database, your storage, your analytics. Pops up, you could do HTTP calls. Basically, you’re writing code that responds to things in Firebase. What this would let you do then is trigger some processing when something update. If someone uploads another cat photo, you could turn around, receive that photo as an event and do something to it, send it off. For

example, there's Google cloud vision APIs. You can send that photo off to cloud vision APIs. Get some information. Get some stats about it and then turn around and update the database from there.

In the past you would've had to bring your own backend. You could have Firebase as a platform with your own backend that does special processing. Now, with cloud functions, you could turn around and provide your own scalable backend that does whatever you wants within the limits cloud functions.

[SPONSOR MESSAGE]

[0:14:06.4] JM: Artificial intelligence is dramatically evolving the way that our world works, and to make AI easier and faster, we need new kinds of hardware and software, which is why Intel acquired Nervana Systems and its platform for deep learning.

Intel Nervana is hiring engineers to help develop a full stack for AI from chip design to software frameworks. Go to softwareengineeringdaily.com/intel to apply for an opening on the team. To learn more about the company, check out the interviews that I've conducted with its engineers. Those are also available at softwareengineeringdaily.com/intel. Come build the future with Intel Nervana. Go to softwareengineeringdaily.com/intel to apply now.

[INTERVIEW CONTINUED]

[0:15:02.7] JM: I want to get into the cloud function stuff a little bit later. I don't know how deep you can go on the inner workings, because I would like to discuss the architecture of Firebase itself a little bit more if we can go deep. Can you describe some of the architecture for how it works? Because I understand that it's kind of this opaque backend and in that's what a developer wants. I'm a developer. I don't really — If I'm writing and deploying software, I don't really want to think about how it's going to scale up to a bunch of users or scale down when those users leave my platform so I'm not overpaying for servers. As a software engineering journalist I am curious what is that architecture look like. What's the multi-tenancy model? What is the sharding and replication that Firebase uses? What kind of stuff can you talk about?

[0:15:51.7] DS: Unfortunately, we can't really say a whole lot about that because Firebase is built on top of Google infrastructure and Google infrastructure is not exactly entirely public as you know. It's hard to make claims about that. For example, Firebase is kind of well-known for not giving you a guarantee about where your data sets like physically in the world because that's just the way Google infrastructure works. Now, practically speaking it's probably going to be in the United States but part of the magic of Firebase is actually built on top of the magic of Google infrastructure. There's quite a few proprietary secrets in there that unfortunately I can't really divulge.

I will say that I think it is pretty well-known that Firebase is actually built on top of MongoDB, MongoDB tends to be sort of the core storage layer. What happens on top of that, this sort of magic that makes it real tiny, that I don't know, and I don't know if I could say if I did know.

[0:16:45.6] JM: Okay. Can you talk about like how — When Firebase was acquired by Google I imagine that the engineering was a lot different. Then since then you've been able to take advantage of some of the economies of scale, like get on Borg and perhaps maybe Google has some really good ways of replicating database or anything, but I guess you really just can't talk about any of that.

[0:17:14.5] DS: Yeah. The problem is that, for me at least, I was hired on as a software engineer and I almost immediately switched over developer advocacy which means I'm now abstracted away from it. I'm more concerned about talking to you and talking to all your listeners and showing up at events. I didn't learn a lot of the infrastructure. There are some things generally the world knows about. We have a special sequel, a scalable SQL like databases. Big table of courses is — Papers have een written about that.

I don't know what Firebase's particular story is as far as its migration from being an independent startup to sort of a Google managed product. I can tell you that I was at a startup called Pulse.io and we're doing application performance monitoring. We were acquired by Google and just from knowing how things were built at Pulse.io and knowing how some things are built to Google, you have to — There's just not overlap. We're choosing some off-the-shelf standard cloud-based services that powers stuff. A lot of people build their stuff on Cloudera, or AWS. Google of course doesn't do that. We have something that has the scale much more. I imagine there were

a lot of growing pains or some challenges getting from one system over to another. I do believe it was good. I don't know how many years it was from the time of Firebase acquisition to the time that the Firebase platform was first offered at io, but it was a significant effort.

[0:18:41.1] JM: Can you talk in broad strokes about the Pulse acquisition? Was that more of an acqui-hire type of situation or was your technology acquired and you had to figure how to migrate that on to Google infrastructure?

[0:18:53.4] DS: I would say it was more of an acqui-hire. I think Google liked what we were doing. We had a small team, there were five of us. We had four engineers and our CEO was more business oriented. They really like the team. They liked what we were doing. They wanted to roll that into the team. A lot of times when you have a team with some expertise that works really well together, it's good to take the entire package rather than trying to find them individually.

My sense that it was the was more of an acqui-hire. If you look at Pulse.io now — In fact, I think if you go to the URL Pulse.io, it just goes to Google now. The product is no longer around. We didn't really have like a transition phase and we told our consumers, "We'll support you for a year, but after that year you're on your own as far as what you're doing for performance monitoring."

[0:19:37.0] JM: Okay. The old domain name migration.

[0:19:40.1] DS: Yaeh. I'm curious. I want to go — It's been a while. I'm curious if Pulse.io actually still just — I think it just goes to regular Google search. It's just like, "Oh, it's gone. It was there and now it's gone." It's a little bit disappointing.

[0:19:53.9] JM: Yeah. We'll talk at the application level. I guess a little bit more on the deep diving type of stuff. When you get the skeptical developers, because there are these skeptical developers that will be like, "Oh, I'm never going to a backend as a service. I'm always going to manage my on premise servers," or maybe if their neck beards a little shorter they'll say, "Oh, yeah. I'll just manage my EC2 cluster."

Do these people ask for certain things, like what are the availability requirements, or how does it do leader election, or what happens if a node dies? Do they ask these kinds of questions and you just can't really answer them? Is there some whitepaper FAQ of guarantees that Firebase gives?

[0:20:36.2] DS: Yeah. I think if you — I can't remember where it's at. If you go to our pricing page, that's when we start to break down exactly what we offer and there is a page we can go to monitor the status and there's an SLA. Now, because were offering a platform as a service, there is no guarantee about things like leader election and those going down. Sometimes we'll name a particular machine cluster that went down and nobody knows what that is other than that name. When that happens, people would jump on Slack and say, "Hey, what happened with this thing?" It's not enough details to know what actually happened.

That said, there is an SLA. Now, it's actually very notable. We released cloud functions in beta which means it doesn't have an SLA but we'd still want people to use it because we think it's great. Then what people are running into is we've kind of have known problems cold startup times. The first time you do deploy function, you executed it, there's this time it takes for the function to actually fire up in a node instance to become available. Everyone wants know, "How do I kill this? How do I drop this cold start?" We really don't have an answer for you. The thing is if you're managing your own cluster you can give yourself a stronger availability guarantee.

Practically speaking, you have a busy app, you'll never really have a cold start problem. You'll have it for a little bit and then your instances will be firing as fast as they need to. That's what everyone wants to know is, "How do I boost the performance? How do I get it faster? How I drop the latency?: Again, in beta, we can't really say anything about that but the hope is that when we do reach the exit beta we'll be able to give stronger guarantees about that. The guarantees would be around the service itself not the internals of the service. Yeah, developers don't want to have to think about these kinds of things but some of them really do and it's kind of like difficult to manage those expectations, I think.

[0:22:22.8] DS: The cold start problem that you're talking about with cloud functions, this is something I discussed with the people I've discussed about AWS Lambda, the serverless Amazon Web services thing. Which is basically, at least this is what they tell me about the

coldstart is, what you're actually doing with a Lambda function or a Google cloud function is you're making a request for the code the that you have. These cloud functions are just like blobs of code. They're stateless code usually and you're making a request for this code to be executed somewhere across a Google cluster or across an Amazon cluster. You don't really care where — When you're making that request it's like the blob of code is being — There's some container that's getting spun up to run that code and once that container is spun up, it's got some time that you stay alive and accept more requests for that same function. If you don't make a request for a certain length of time, then the container just gets spun down.

The reason these functions are so cheap to purchase and the reason the world is going towards this serverless model is because Google or Amazon, they both have these giant economies of scale. They just have like servers that are sort of lying around and they can't make good guarantees about what are these servers capable of doing but they can match code to available servers and execute the code on the server. It takes some time to do the matching or do the spinning up or pairing up the resources. Is that accurate? Is it an accurate description of the cold start problem?

[0:24:01.4] DS: Yeah, I would say that that's it in a nutshell. Yeah, Google does have — I don't even know the numbers. If I did, I don't know if I can say. There are lots and lots of servers out there sort of sitting around waiting for work. A given server maybe even know what kind of work it's going to receive, but it's going to receive work. The first time you or a client wants to invoke a function, something similar has to be running at some point, but it's not we're not going to start it until it's necessary. Nothing is going to be running until the first request comes in. When that first request comes we'll start up a node instance, load of the JavaScript, hand it the event, and it'll execute, and then we're going to kind of make the assumption at that point that another request could come in but we don't know when, we don't know where but will keep that instance running for a while. It's warm at that point. Then the second and third indications will continue.

It is just like you said, after some amount of time, after the last function has been executed, it doesn't make sense to keep it running anymore so we'll just, like you said, drop it. That is the economy of scale. We're only running ourselves what we think is needed. Whereas if you're running your own clusters, you may have to do some sort of manual scaling or you may have to do — You have to be intelligent or you may have to you manually commission or decommission

server. With Google's infrastructure, all that happens completely automatically when you go serverless.

[0:25:19.2] JM: Let's talk more about that serverless idea because the shows I did about Lambda, AWS Lambda. I haven't done anything about GCP or Google cloud functions. I should do those. The shows I've done about Lambda, there is this statelessness. Since you're just getting this ephemeral container, you don't have a whole lot of say about what happens after you compute a result and get that result back. I think the synergy between Google cloud functions and Firebase is that Firebase is your system of record and the Google cloud functions are the compute that runs on top of it.

[0:26:01.5] DS: Yeah, that's exactly the case. We have, I think on out GitHub repo, some 30 some samples that show how cloud functions essentially glues together the different Firebase features. Firebase real-time databases, your persistence layer, cloud storage is your binary file object blob space. We have cloud messaging's. If you want to send notifications to clients, you can use that. We also Pub/Sub if your app uses that, and also HTTP functions, and also analytics. You can actually respond to analytics events as well.

Firebase real-time database is the persistence layer. If you have something that's stateful, you're inevitably going to have to query the database during every function of execution. If you're the kind of developer that likes to think about, "Oh, I want to have like memchache sitting in my function so I don't have to query the database every time." You could try that but there's no guarantee that there will be any state left over between function and executions. We just can't guarantee that that's the only way it can scale. If there was state, we'd have to sort of solve that as a feature as well. Yeah, real-time database is the feature for state.

It turns out I'm actually doing a talk at I/O 2017 on exactly that topic. What I did was I wrote a turn-based multiplayer game on top of Firebase using cloud functions. Of course games have state. That's the very definition of a game is that at any given time there are pieces on a table. There is state. How do you manage that? Well, you have to get a little bit creative and come up with a system for using cloud functions to essentially advance the state of the game programmatically by reading and writing the database. If it's turn-based, that's even easier because you can set up all the rules for how the state machine advances. If you don't have a

state machine, if it's real time, then you have a little more challenge and trying to come up with transactions that make sure that multiple players are competing for resources correctly. Yeah, cloud functions actually opens up a lot of what could not have been done but you have to get creative when it comes to maintaining state. Firebase does offer that.

[0:27:59.8] JM: If I was building chess, for example. You just said turn-based game. If I was building chess using Firebase and cloud functions, I open up my chess.com webpage which has some HTML or JavaScript code that makes requests for resources from the Firebase endpoints, do I need to use cloud functions for anything there? What would I use cloud functions for in that application?

[0:28:31.6] DS: Probably what you'd want to do is — You could actually, I think, get away with not using cloud functions. You just have a hard time making sure that players don't cheat. Think about this, if you have two clients both program with the rules of chess running in different computers, those clients have to agree with each other on how to play the game correctly. Maybe the client logic — Say, it's a web app and users interact with the webpage to sort of indicate their intent on where to move. You're kind of depending on the JavaScript that gets run to check to make sure whether not a move is valid, whether or not the game is actually over.

With web apps, know you can inject whatever JavaScript you want running in the browser at any time. There's no protections on sort of the veracity of the code. If your game is highly dependent on not trusting your clients to cheat or maybe have a tournaments system where there's maybe cash money rewards for doing well, you probably don't want to do that in the client. You probably want to push that logic into the server. That's exactly what cloud functions does. Instead of depending on the client to observe the rules of the game, you actually ask the server to observe the rules the games.

The clients, instead of reading and writing the database, or reading and writing the chess table contents directly, you would send a command to the server and say, "Server, this is what I want to do. Would you execute that formula?" Then the server is free to turn around and say, "Oh, you totally violate the rules of a rook move. Don't do that." For the clients to agree if someone won, the server can actually decide that for you. It will know if there's a checkmate situation and simply in the game in a checkmate rather than having the players to agree to in the game.

You're making those decisions on the server using a cloud function rather than logic that's baked into the app, and that's one of the big, or what I've always said, is one of the big selling points of cloud functions, is your logic is safe and secure. That's no one's going to deploy a cloud function overtop of yours unless they somehow get your password, but clients are far less secure.

[0:30:33.5] JM: I have an application that I'm building with the team right now, it's a photo sharing applications. It's like a social network sort of thing. We started off with this node and Mongo backend, typical three tier architecture. The node and Mongo backend with an iOS or web front end. Recently we actually added Firebase just to get chat up and running, because we wanted to get chat and we're like, "Yeah, rather than program this in node, let's just get Firebase up and running in it'd give us a chance to try Firebase." It was awesome.

We're starting to consider putting all of our new backend functionality into Firebase, maybe even deprecating the node backend altogether. Really not sure, but I have gotten advice from several people who have just said, "Oh, if you can, go completely Firebase." If the cost structure makes sense at least from your business point of view. If you're building something that's a total commodity, you probably would not want to be on Firebase because your cost structure is really important, but a lot of tech companies are really differentiated. Particularly like a social network, the margins are pretty good, so you might as well be on something that minimizes the amount of work you have to do like Firebase.

I do know. What kinds of functionality are still hard to do between Firebase and cloud functions? What should I know before I go whole hog into deprecating my node backend and pushing everything into Firebase?

[0:32:05.5] DS: We get that question a lot actually. To boil it down is what should I not use cloud functions for, and it's a fair question. I'll tell you briefly what people are using it for. If you know there is a piece of software that we wrote called Firebase Q, and it runs in a node container that you control. You basically write into a location of the database and then Firebase Q processes those commands out and then does little bits of work. That's very close to what cloud functions does.

A lot of people are migrating from Firebase Q to cloud functions. That's a very simple and straightforward path, or if you just run like an HTTP server you have endpoints that you want to access, very easy to move that into cloud functions.

With things start to break down for some kinds of apps is if you have very high intensity CPU compute operations to do. Transcoding video is an example, or if you have hours and hours of video, cloud function is probably not work for you because we limit the execution time of the of the function to nine minutes, I believe. Normally, I think the default is one minute and you can bump it up to nine if you need it. Nine minutes, quite frankly, is not enough to transcode videos, so those apps would not be able to port.

If you do have any sense of state in your app that can't be represented by database reads and writes, if you need everyone to go to a particular server and that server manage state, you probably couldn't use cloud functions for that.

What else? Anything that has probably really high I/O requirements — I don't know. That might not be very great as well, because you are paying for outbound networking. If you can get that cheaper somewhere else, you might opt for that simply because the cost structure for that kind of application might be more effectively done on your own managed servers or in some other managed networks. Those are few of things off the top of my head I think that would not work for people.

[0:33:56.1] JM: There have been a lot of listeners who have asked about Google's cloud strategy and I'd like to explore that with you. It seems to me that we're moving towards a world where multi-cloud becomes easier. The hardest parts about integrating clouds are things like load-balancing and networking and all of that stuff is getting abstracted away, so we're just going to be using APIs and backend as service things. Do you think it's becoming easier to be multi-cloud?

[0:34:33.1] DS: I'm actually not to certain what you mean by multi-cloud.

[0:34:37.1] JM: I can use Google services together with AWS services, or Twilio, or let's say DigitalOcean comes out with some platform as a service thing that I want to use. There's less of this, like, "Are you on AWS? Are you on Microsoft? Are you on Google?" It's more of this, "What service are you using?" You just have a buffet of options that play nicely together.

[0:35:01.7] DS: I see. That issue is, "Could you build your application or your business on top of what amounts to, not just one or the other but every or as much as you need or as little as you need from all these different options?"

[0:35:01.7] JM: That's right.

[0:35:16.4] DS: Okay. Yeah. I don't know. Yeah, it's a good question. I actually haven't thought about that too much. What you're describing sounds like — I did a lot of web development back in the 2000s and at one point I think it was like 2004, the term a web mashup, became popular where you didn't really build the entire web app you just called out to all these different things and then merge them altogether in your web app.

For example, you could pull a map from Google and you could call some APIs to get, say, transit stops on that and that up on there then pull other like — Just information from all around and put it on all one webpage. I don't see any reason why people would not be able to build application applications similarly on top of different cloud offering. If Google offers you something that's great and you can't get it elsewhere but you have a another part of your app could be well served somewhere else, then why not. I think it's also kind of related to the idea of micro-services where you don't have everything under one big infrastructure. What you do is you solve individual problems in individual infrastructures and let them solve their problems individually very well and maybe tie a few things together from a single data storage provider. That seems very reasonable to me.

Honestly, I don't know how many people are doing that but we could move in that direction where some companies are highly specialized and some kind of platform feature and others just are not and that's fine. You pick and choose from all of them.

[SPONSOR MESSAGE]

[0:36:46.4] JM: For years, when I started building a new app, I would use MongoDB. Now, I use MongoDB Atlas. MongoDB Atlas is the easiest way to use MongoDB in the cloud. It's never been easier to hit the ground running. MongoDB Atlas is the only database as a service from the engineers who built MongoDB. The dashboard is simple and intuitive, but it provides all the functionality that you need. The customer service is staffed by people who can respond to your technical questions about Mongo.

With continuous back-up, VPC peering, monitoring, and security features, MongoDB Atlas gives you everything you need from MongoDB in an easy-to-use service. You could forget about needing to patch your Mongo instances and keep it up-to-date, because Atlas automatically updates its version. Check you mongodb.com/sedaily to get started with MongoDB Atlas and get $10 credit for free. Even if you're already running MongoDB in the cloud, Atlas makes migrating your deployment from another cloud service provider trivial with its live import feature.

Get started with a free three-node replica set, no credit card is required. As an inclusive offer for Software Engineering Daily listeners, use code "sedaily" for $10 credit when you're ready to scale up. Go to mongodb.com/sedaily to check it out. Thanks to MongoDB for being a repeat sponsor of Software Engineering Daily. It means a whole lot to us.

[INTERVIEW CONTINUED]

[0:38:47.3] JM: You talked about what we might use cloud functions for in a board game, for example, where you've got your client application. It's maybe in React or Angular or something, but you don't want to put too much business logic in it. If you're using Firebase as your system of record, you might want cloud functions to serve as business logic, just this business logic that's kind stateless, it will enforce certain requirements on your app regardless. There're other types of processing that we might want to do on top of our application. We could want to be doing some machine learning, or just some dataflow, like stuff in cloud dataflow where we're just shuttling data from one place to another.

What I want to run these kinds of — Maybe you could talk about what kinds of processing — When do I start thinking about, “Okay. I need to use your TensorFlow, or a cloud dataflow for certain processing aspects rather than just using these cloud functions.”

[0:39:52.7] DS: Yeah. Cloud functions is, for many apps, your first best stop for typical units of business logic. Obviously, as you say that that breaks down when you start doing computationally intense things like you know TensorFlow, that's why we have offerings like compute engines. If you really need to get in there, do some heavy computing, you can have a compute engine instance and do whatever you need there. That doesn't really make sense to try to try to implement in cloud functions. Maybe cloud functions would end up calling into maybe a service that you provide on this compute engine backend and they may relay that back to the client or send that onto the next thing. You might think of cloud functions as kind of a traffic cop directing other things.

Yeah, for really computationally intense stuff, you do want to go to something that's more appropriate. Now, I've never tried to build anything like that, so I'm just sort of supposing what developers might do. I can certainly see that to be the case where you end up sort of mixing and matching and choosing. Firebase is actually kind of a good example of that. Firebase is great for app developers. It’s great when you start out. It will scale with you, but at some point your app, if it becomes successful, will outgrow maybe some of Firebase's offerings and that's a great problem to have. You may want to scale into something bigger and Google does provide that.

We think of Firebase as your on-ramp onto app development. You want to get started. You want to get going quickly. You know that it's going to meet your short-term needs which are not confident. It will meet your super long term needs. We are definitely saying that if you go with Firebase, we will try to help you ease into that enterprise level or that sort of — The more highly intense application backends. They'll just do more things for you than you could ever do with Firebase, and that's a real thing. That's part of our strategy.

[0:41:38.6] JM: Firebase will say, “Hey, we’re going to provide the availability you need, the consistency you need for the general application. We’re not going to give you scientific level requirements or scientific level guarantees,” but if you want that kind of thing, which you’ll

probably only want at scale. When you have a lot of customers, you can go to something like Cloud Spanner or I think there's these other databases, Hosted, Bigtable.

Do you know what — What is that situation look like? Is it a price thing? Is it an availability sort of thing? When do you start to consider other databases than Firebase?

[0:42:21.3] DS: The big observation about real-time database is that it's really easy to work with and it's real-time, but the querying options are rather limited, so you can't you can't effectively have multiple WHERE clauses. You can't say, “Of all of these collection of things, I only want some where X equals two or X equals three.” You have to pick one or the other. You can't do both. You can only order and filter certain ways, and that's very limiting for a lot of applications. It's when you start to hit those limits that you think about upgrading to something different.

Now, in that particular case, simply because you can't do multiple WHERE clauses doesn't mean a Firebase isn’t suitable. You can architect your app to duplicate data in such a way that those quarries do effectively become possible and just you just have to think harder about how you want to organize your data. If that becomes too unwieldy, if you're duplicating too much logic or if it's just not fitting the sort of ad hoc query requirements of your app then you would move to something else. For every app, that's going to be different. It's a matter of product requirement versus engineering availability. If the engineering availability or capability of Firebase isn’t matching what you need, then you move on to those other options.

[0:43:28.7] JM: Why is it that it's difficult to do stuff like multiple WHERE clauses?

[0:43:33.8] DS: That's the thing. It's not really difficult, per se. It's difficult to do it in a real-time capacity. It’s trade off of one thing versus another thing. If you have a SQL database, those are super easy to work with. You can normalize all your data and it gives you the flexibility to join between multiple tables with a lot of flexibility and creativity. That's very easy. Have you ever tried to serve real-time content out of a SQL database? Well, I think for most people, they probably haven't tried. How do you even make that happen? I don't know.

When you go if the NoSQL option we have, better ability to scale. You have better ability to focus data processing in a way that would be seen as very simplistic and SQL, but also very

scalable in a way that it can't do with SQL. I don't think it's necessarily a matter of difficulty so much as it is a trade-off that you're giving.

The thing is like I think James Tamplin, one of the cofounders of Firebase, sent a tweet out recently asking or soliciting for participation in a program where you get to experiment with enhanced querying capabilities of Firebase. Those are injuring problems that we can certainly tackle. They're just not the easiest ones. To me, it boils down to the differences between SQL and NoSQL.

[0:44:49.6] JM: There must be people who you've talked to who they've gone through this where they've on-boarded with Firebase, they scaled to a certain point and started to say, "Okay. We need database X, or we need database Y, or we need Cassandra for this thing." What is the pattern that you see? Is there some kind of offline data that they've moved to these other databases, or what patterns do you see?

[0:45:10.9] DS: This comes up from time to time on a variety of venues. The generalized question is; does it ever make sense for me to use Firebase along with a SQL database or whatever other options? For some people, that's not — Think about, "Oh, you could use both at the same time, or if I could, how could I?"

That's actually gotten a lot easier with cloud functions. In the past, people use Firebase queues, essentially monitor, updates a channel of updates and then distribute that to other databases. You can do the exact same thing with cloud functions. You can say, "Whenever data changes in a certain place, we'll just copy that out to another location." The ideas that if real-time database is your first line of defense against reads and writes, you can essentially duplicate that out to other databases.

The trick is; well, if a change happens in one of these outside databases, how do you get that back in a Firebase? That's a different problem. Cloud functions can't really solve that for you, but we're finding that some people might try to get the best of both worlds by duplicating between a backend of their — Or some other database of their choosing and Firebase.

[0:46:10.9] JM: Another exciting backend project that we've been covering recent on Software Engineering Daily is GraphQL, which is a middleware server. It federates requests to a variety of data sources based on what the client needs. The way I see it is where Firebase helps you simplify a brand-new backend that hasn't been created yet — Like you said, it's an on-ramp. GraphQL has been shown to be useful for making backend that already exists and they have a collection of different data sources. It unifies these data sources and it makes requesting data from a variety of data sources less complex. How do you compare the two projects? Have you looked much GraphQL?

[0:46:58.1] DS: I have not. Just the way you're describing it though, makes it sound like it would be extremely convenient for certain applications that do need these multiple backend databases each in their own power, but unified through a single interface.

[0:47:12.0] JM: Exactly. Yeah, that's exactly what it does. Yeah, okay. My next question was going to be like have you seen any people using GraphQL and Firebase together? Because s I find that a compelling options because I could see certain applications would have real-time requirements and other applications do not. Maybe that it's just some sort of off-line data processing thing and if you're using GraphQL, then you can have a unified interface over which you request things and the GraphQL server could translate those requests into something that Firebase can understand or something that Cassandra could understand.

[0:47:52.6] DS: I'm not sure if GraphQL would add a layer of functionality on top of Firebase in a way that makes sense for someone who is choosing Firebase. I'll say a little bit more about. The whole thing about Firebase is the real-time nature. It's NoSQL database, but you can treat it like a basic NoSQL database and if you don't need its real-time features, then fine. You don't have to use it, but that doesn't make Firebase the most useful offering on that front. It sounds like if using GraphQL, you don't need a real-time stuff, you just need the aggregation capabilities. You could certainly put Firebase behind that if there is a driver that lets you issue quarries the Firebase.

I don't know what it would be adding other than the ability to augment data from Firebase along with some of these other sources. You'd lose its real-time nature. I'm assuming GraphQL isn't something that's going to ship out data from Firebase to clients because it knows in advance

what they wanted or they're listening to it. I see it as it could certainly sit on top of it, but I don't think it would necessary adding anything to that offering.

[0:48:56.5] JM: Yeah, interesting. Maybe that's a different show. If somebody out there has done interesting stuff with GraphQL and Firebase. Okay, so I knew we're wrapping near the end of time. There are going to be a bunch of presentations about Firebase at Google I/O which I am attending. What some of the stuff that's I should look forward to and potentially cover?

[0:49:17.4] DS: There's going to be so many things. Actually, there's a blog post coming out about that. I think your question, beat that punch by a few days. We're going to have new stuff announcements. Firebase is always releasing new things. Actually, that was the topic of one of my talks that I gave in Boston, Chicago; what's new and Firebase over the past year. We're constantly releasing new things. There's going to be new things at I/O.

I think there's going to be some talks about app quality. One of the things I'm particularly interested in is how Firebase can help the overall quality of your app, and it's a subjective thing; what is that quality? Well, it's, of course, things like crashes and testing and stuff like that. I'm kind of excited to see Firebase be used more than just the backend, like the development stuff.

There's always swelling interest around analytics. Like analytics is huge for a lot of companies and we've seen that a lot of apps build five or six different SDKs into their app. We've heard loud and clear like analytics is pretty big deal. I think there's going to be some talk around analytics. Like I said, my talk is just going to talk about how to use it for very practical things. How do you put the rubber on the road and actually use cloud's functions in a way that will serve your app and not just be kind of a cool thing.

I think the I/O schedule is already up. The Firebase track should already be published, but I would encourage people to go check that out, maybe plan out there day if they're going to I/O. If anything's not on there that you're interested in, the Firebase team will be at a kiosk or a tent. We'll be there. I'll probably be there most the time so you can come and talk to me and say hi.

[0:50:49.2] JM: Cool. I will.

[0:50:50.2] DS: Absolutely. Yeah. We love the talk to people. I love to talk to people. I'm actually kind of an introvert, but I love it when people want to come up and talk about Firebase because it's, of course, interesting to me. Definitely stop by and say hi.

[0:51:02.5] JM: Great. Doug, I want to thank you for coming on Software Engineering Daily. It's been a pleasure talking to you and it's been a pleasure working with Firebase since we started building it in our application.

[0:51:11.5] DS: That makes me very happy. Nothing makes me happier to know that developers are having their needs met by Firebase, because that's exactly what we're here to do. That is our mission, is to help you make successful apps.

[0:51:21.5] JM: It's happening. It's a great product. Rally love it.

[0:51:25.7] DS: Yeah, that's great, and thank you for having me on the show. I really appreciate the opportunity to talk with you.

[END OF INTERVIEW]

[0:51:32.4] JM: Deep learning is at the forefront of evolving computing and promises to dramatically improve how our world works. In order to get us to that bright future, we need new kinds of hardware and new interfaces between this AI hardware and the higher level software. That's why Intel acquired Nervana Systems, a platform for deep learning.

Intel Nervana is hiring engineers to help develop this full stack for AI, from chip design to software frameworks. Go to softwareengineeringdaily.com/intel to apply for a job at Intel Nervana. If you don't know much about Intel Nervana, you can check out the interviews that I've conducted with engineers from Intel Nervana, and those are available at softwareengineering.com/intel as well.

Come build the future of AI and deep learning at Intel Nervana. Go to softwareengineering.com/intel and apply to work at Intel Nervana. Thanks to Intel Nervana for being a sponsor of

Software Engineering Daily, and I really enjoyed the interviews I have done with the Intel Nervana staff. I think you'll enjoy them too.

[END]