EPISODE 1010

[INTRODUCTION]

[00:00:00] JM: Imagine yourself using a gig economy application like Uber. That application generates lots of notifications. There is SMS, tere is a mobile phone application native updates, there's emails. If you order a ride from Uber, you might receive a text message and a push notification at the same time, and if an app overloads the user with notifications, the user might end up annoyed and delete the app from their phone altogether. But perhaps all of these notifications are necessary. You would rather get three simultaneous notifications from your food delivery app than fail to get your food on time.

If you're a mobile application developer, you might not know which channels are working for you. You might not know which SMS notifications are reaching the user and which times the user needs to get an email update. How do you solve this problem? What choice do you have but to send the user all these different kinds of notification? At large companies like LinkedIn or like Uber, there are actually entire teams devoted to figuring out how to optimize the notifications that they send you. It has a surprisingly large impact on the usability of a mobile application.

Troy Goode is the founder of Courier, a company that provides this kind of notification optimization as a service, and this might sound like a small trivial problem, but it actually has a large impact on the usage of our apps and it's not an easy engineering problem. Troy joins the show to talk about the problem that Courier solves and the backend infrastructure that powers it. Courier is built entirely on serverless APIs. This is a great case study in how to build a completely scalable infrastructure product based on serverless tools, and we've done many other shows on serverless products and things that have been built entirely in serverless. If you want to find all those episodes, you can go to softwaredaily.com. You can look at our mobile apps. They have all of our episodes organized by categories and in terms of which episodes are the most popular. You can check those out at softwaredaily.com.

[SPONSOR MESSAGE]

[00:02:19] JM: This episode of Software Engineering Daily is brought to you by Datadog, a full stack monitoring platform that integrates with over 350 technologies like Gremlin, PagerDuty, AWS Lambda, Spinnaker and more. With rich visualizations and algorithmic alerts, Datadog can help you monitor the effects of chaos experiments. It can also identify weaknesses and improve the reliability of your systems. Visit softwareengineeringdaily.com/datadog to start a free 14-day trial and receive one of Datadog's famously cozy t-shirts. That's softwareengineeringdaily.com/datadog.

Thank you to Datadog for being a long-running sponsor of Software Engineering Daily.

[INTERVIEW]

[00:03:14] JM: Troy Goode, welcome to Software Engineering Daily.

[00:03:16] TG: Thanks, Jeff. I'm glad to be here.

[00:03:18] JM: Whenever I use a gig economy service, I get lots of notifications.

[00:03:23] TG: Absolutely.

[00:03:24] JM: This is true whether I'm using grocery delivery, or I'm using ridesharing, I get SMS, I get mobile phone updates, I get emails and sometimes I get so many different types of notifications that I end up silencing all of them and then I get a phone call because the service can't reach me any other way.

[00:03:44] TG: That's right.

[00:03:46] JM: How did we get to the point where we are drowning in notifications?

[00:03:51] TG: I think that if we rewind the clock maybe even only 10 years, there was only really one meaningful notification channel, and that was email for most software products. I think that what has happened with the advent of mobile devices and the ubiquity of them is now it's so easy for technology companies to reach you, and frankly those notification channels

perform so much higher than email in terms of creating an engagement event. Basically, you clicking on that and driving you back into their product that it's a bit of a tragedy of the commons situation where it's optimal for every one of those individual products to maximize the likelihood that you're going to engage by reaching out to you with many notifications.

On the other hand, when everybody does that, that creates a culture where you become very fatigued by notifications. One where you then start to think about unsubscribing, and I typically block badges because that drives my OCD crazy. All of a sudden, because everybody's doing it, now it's maybe not the optimal strategy for any one of them, but who's going to be the first one to break and stop sending as many? Because that's not going to cause the rest of the industry to do the same.

[00:05:05] JM: Let's say I'm building an application has a lot of notifications, what are some anti-patterns of people who are building those kinds of apps?

[00:05:16] TG: I think that easily the number anti-pattern is if you are a product – If you have a product that supports multiple channels, so let's think about typically it would be maybe email, SMS, voice, push. Those would be typical channels might support. Sending notifications across all of them, that's probably the biggest anti-pattern and one of the biggest ways to drive unsubscribes. Certainly, there are scenarios, maybe like emergency scenarios where that might make sense. If you're PagerDuty, there are scenarios where this makes sense. But for most products, you need to be more selective about which channel you're using and when.

[00:05:50] JM: Right. So, like grocery delivery, there's a few apps where every time the groceries are going to be delivered, I get a message on SMS and in the app, and that's actually – I don't know if it's bad, because I do want to know that they're being delivered, but it seems like maybe it varies from person to person, from customer to customer, from grocery delivery to grocery delivery.

Are there systems for – And I know you're working on a system like this, and this is the kind of use case that drove you to start Courier. But prior to Courier, what were people doing to customize their notification systems?

[00:06:37] TG: I mean, even today, we're still in the early days. I would say most firms are just building this out internally. There's one I can talk about, or there's a couple that I can talk about because they've talked about what they're doing publicly. One of them is Twitch, so that Amazon-owned streaming company. They, the year before last, launched what they call Smart Notifications.

Prior to Smart Notifications, they had three channels. They had in-app notifications that appear while you're in the Twitch experience either on the web or in your mobile app. They had Mobile Push notifications that would show up to the home screen of your mobile device when you weren't using the app, and they had email notifications.

They were sending – Let's say I follow your Twitch stream and you come online. They want to let me know that because it's their business model to drive me inside of the app. I'm going to see advertisements, and that's how Twitch is going to drive revenue.

If I don't know that you've gone online, then there is no value being produced. It's not YouTube where I can come and check it out later, really. It's oriented around this live event and that notification is incredibly important.

They started out by sending notifications to all three of those channels at the same time. Boom! Boom! Boom! We've experienced this, of course, your delivery apps and other apps. What they found was the unsubscribe rate skyrocketed. When somebody would unsubscribe, now it wasn't they were unsubscribing from one of those channels. They were unsubscribing from all of them.

All of a sudden, now you can't reach them at all. The next time Jeff comes online to start streaming, because Troy unsubscribed the prior time, they can't reach Troy. Troy doesn't come back to the platform potentially ever. There's no other way to reach him, and now you've lost that revenue.

[00:08:23] JM: And what would be the alternative to that? How could twitch avoid the scenario of somebody completely unsubscribing from those notifications?

[00:08:37] TG: Yes. Their smart notifications platform that they released, what it does pretty simply is it keeps those three channels in place, but it just adds some logic around that, and they have other things that they do that's more sophisticated. But at the simplest terms, what it says is, “Okay. We’re going to prioritize these channels based upon the likelihood that it’s going to cause Troy to engage and come into our experience.”

We know, from benchmark data, that the highest performing channel that you can have is in-app notifications if you have long session times. Twitch has long session times. People come, they watch. They stay.

Let's say I'm watching somebody else's stream, and now Jeff comes online and I followed Jeff. If I'm online in that experience, then Twitch will show me an in-app notification while I'm in that experience inside of Twitch and they will not send that Mobile Push notification and they will not send that email, and that's going to be the highest likelihood activity to get me to reengage and stay longer within Twitch.

But let’s say I'm not. I'm not online. I'm not watching Twitch. I'm walking down the street. I'm at lunch. Then they look and see, “Okay. Has Troy downloaded the mobile app? Has he accepted push notifications? Has he engaged with the mobile experience within the last X-period of time to consider him an active mobile user? If so, now, we send the Mobile Push.” If none of those are true, then finally use email as that fallback mechanism. It's the worst performing channel, but it is ubiquitous and at least it’s a last-ditch way to try and reach me.

[00:10:04] JM: Before we get further into the mechanics and the details of this, let's just drive this home a little bit further, because Courier is pretty beautiful. You just showed to me and you’re spending a lot of time on this, and I know that there's people listening right now who are like, “This is definitely not something I'm interested in. Why would I care about notification optimization? How can this really be that important?” But as I've understood from looking at your product and just talking to you, this is actually something that impacts a ton of applications.

[00:10:41] TG: That’s right.

[00:10:41] JM: Can you just give more of a survey of the landscape and explain why maybe you call it notification optimization or whatever you want to call this category of problem. Why is this important?

[00:10:54] TG: Yeah. I think that it is important to any product whether you're a technology company or some other service where what you need to do is drive your customers and users back into your software experience, right?

A lot of technology companies fit this bill, right? B2B, SaaS products, absolutely. A lot of B2C products. Media products for sure, but also maybe more traditional services. We think about things like United Airlines and all the notifications that we get often on many channels at the same time about a gate change. They are trying to drive us in for more information into their app so that we can be informed about what has happened.

Fundamentally, businesses usually need to reach and communicate with their users, and the way you do so via technology today is through notifications. Whether that'd be email, or Mobile Push, or one of the many other channels.

What has happened though over especially the last 4 to 5 years is that the number of channels that are available to us as engineers and product managers have exploded, but the complexity of integrating them is incredibly high. What frequently happens is in the earliest stages of a software product, we just stick to email or we just stick to Mobile Push.

If you're just in the Mobile Push landscape, so let's say you have a mobile first product, that might be pretty good for you actually, because Mobile Push is a very strongly performing channel. I think there's ways that Courier can add value there, but where we had the most value is if you're starting on the web and you starting with just email, because email is the worst performing channel. If you're doing well at email, you can get about a 3% to 5% engagement rate for clickthroughs. If you add another channel, any user you interact with on that channel, you can expect in the 20% to 30% range of clickthrough. You're looking at 5X or more increased engagement rates just by adding an additional channel.

Now the question comes, “How easily can I add that channel and how easily can I maintain it overtime, because as I add more channels, I have more templates, more content, more copy that needs to be created and controlled.”

[00:13:00] JM: As a company goes from email and SMS, to email and SMS and perhaps intercom, what happens internally at the company as they're getting more and more messaging-based systems? Maybe they’re adding Twilio, maybe some kind of Slack integration. They’re adding Intercom. Who inside the company is managing all those notification flows? How is the company keeping track of, “Okay. This happens and then in tees-off a text message, or this happens and then it opens up a Zendesk ticket and it's going to send them an email and open up a Zendesk ticket over email.” Is there somebody inside the company that is the central point of understanding the user experience of notifications?

[00:13:57] TG: It’s a really interesting question. In the earliest days, it ends up being the engineers. Typically, if you're a small company, the product manager is probably who’s making some of these decisions, because the product manager is the only person incentivized to really be thinking about engagement.

Later stage companies have growth teams, and those growth teams maybe take a little bit of that over. But in the earliest days, the product manager wants this, but the engineers are the only one that can implement it. Look, if you’re just going to add email, it’s simple. Just sign up for a SendGrid account, or Mailgun, Postmark, they have really simple UIs and APIs to be able to connect that into your platform. You add that second channel. You add SMS. Now all of a sudden you have to either do what we talked about not doing, which is just send it to both of those APIs and plug in both APIs and just send it both, or you're putting some sort of control layer on top of that, and it could be as simple as a switch statement or an if/then. But you'll start to think about, “Okay. Well, what are the conditions that are going to control this if/then statement?” Is it going to use context? Things like is the user online, which means you need to have a presence system? Is it going to use historical data about which of these channels the user has engaged in previously or preference data. So you’ve got to start to pool from a persistence letter to inform this decision. You start to do things like, “Okay. Well, I have enough then, but what happens if I’ve picked one of those channels and deliver to that channel fails?” That API is offline. Well, do I queue that back up to deliver it on that channel later or do I fail

over to a different channel? It might depend on notification and the urgency of that notification. Some you might be okay with it that notification being delivered tomorrow. Others you might say, “Look, I just want to get it out on any of these channels immediately.”

There's a lot of this complexity that the engineer has to build out, but the engineer honestly often doesn't care, right? It's the product manager who cares. The engineer just wants to get back to building value inside of the core product and they kind of just get pulled aside, and we see in the largest organizations, they have full dedicated teams and departments that literally work on this kind of infrastructure all day, every day for years.

[00:16:13] JM: Can you say more about that?

[00:16:14] TG: Absolutely. There’re a couple public examples I'm happy to share. LinkedIn has an internal infrastructure named Air Traffic Controller and it's available online on their blog. You can go check it out. They talk about the billions of notifications that they send every day. All of the LinkedIn teams across the world, across their campus down here, they all use the same internal notification infrastructure. So they're all kind of calling this internal API.

There are 30 engineers that work full time on that infrastructure all day, every day and have done so for years. So you're talking tens of millions of dollars of investment into a control plane. They can understand should I send this notification to Jeff over Mobile Push or over email? Should I send this notification to Jeff at all? Should I send it batched with other updates?

If somebody looked at your LinkedIn profile, do we send you a notification immediately or do we wait and say, “37 people looked at your profile, Jeff,” and include that as a different notification. When do we send it? If somebody looked at your profile or sent you a message at 3 AM, do we hit you up immediately or do we wait until 9 AM your local time and deliver that notification? All of that infrastructure that makes these decisions is abstracted away from the core product engineers into this air traffic control system that is, again, built by 30+ engineers within LinkedIn and provide it as a service to the rest of the organization.

[00:17:41] JM: The idea of Courier is that kind of air traffic controller, this kind of person who is deciding the user flows of receiving notifications, interacting with those notifications. Courier does that as a service. Can you explain what your company builds in more detail?

[00:18:06] TG: Yeah, sure. Our YC pitch was segment for notifications. The core idea originally, and I think still, is take all of the channels that you're maybe already using or aspire to use and the actual providers for them. We don't send SMS. We don't send email. Our customers plug in their Twilio account, their SendGrid account, their Slack bot. They plug those into Courier and then we offer a single API. Our customers and developers publish an event to the Courier API and then we use that context and those rules in the benchmark data to figure out which of those channels should we deliver that notification to Jeff on right now.

In addition to that, one of our key value propositions and something that's kind of just ingrained in our culture is enabling non-engineers. So taking the work that engineers do and helping them to empower their other teammates so that somebody who is closer to the problem, that product manager or that growth team member, they can take ownership over these questions and these challenges. They can add new channels. They can change the content and the copy of these notifications without filing a JIRA ticket saying, "Hey, Troy. Can you change the text on these 20 templates? Choose from X to Y."

[00:19:21] JM: Do the tools like Twilio or SendGrid or whatever else is being used to manage notifications? Do any of them have workflow managers like this or is this just this is a piece of middleware that nobody has built as far as you know?

[00:19:38] TG: The marketing side of the world has had this forever, marketing automations. My background, I led engineering for a company called Eloqua, which helped create the marketing automation space. We took that public back in 2012 and ultimately sold it to Oracle. If you're sending marketing campaigns, email marketing campaigns or even today multichannel marketing campaigns, you have infrastructure. You have middleware to do this, but that's really about promotional activities. It's about scheduling campaigns that are going to go out at certain times. It's about then responding to out of product events, things like did you engage – Did you fill out this gated entry form to download this whitepaper? If so, we're going to put you into this separate drip campaign and follow-up on this cadence.

What we haven't had is the equivalent of that for product businesses, for triggered events that are happening and coming out of your product. Some get close, things like customer.I/O, and intercom can do aspects of this, but they are still more focused on those marketing use cases rather than the driving internal product engagement use cases that product management and growth teams care about.

[00:20:40] JM: In that example you gave with Twitch. So like I get a Twitch stratification, and I open that notification. That, I can understand getting feedback or the feedback being driven back into Twitch and Twitch being able to make decisions based off of when that user open the notification and how they interacted with it. Is that true for all these channels? Can I get feedback on an in-app notification, as well as an email, as well as an SMS? Do all of these things give me enough information to be able to have insights about what a particular user prefers?

[00:21:23] TG: The answer is mostly yes, and that while the method varies, each of those channels have ways that you can access that engagement data. Fundamentally, did the user engage? Now, email is the oldest and most mature of these channels and it has the most options available sometimes to the chagrin of users. We've seen superhuman have some controversy over their open rate tracking, right? That's been a feature of email for a long time. Do people open this? You can't generally tell that in many other channels, but you can always, with different methods, tell, “Did the user click?” A good notification has a call to action. It's not an informational FYI. It's, “Hey, Jeff. I'd like you to do something. I want you to login and watch this Twitch stream. I want you – Troy@ mentioned you. I want you to reply to him or at least look at what he said. You put stuff into your shopping cart two weeks ago and it’s still sitting there. Do you want to complete that purchase? There’s something that you're asking that user to do.

You can tie the action of did they perform that action or at least take the next step toward that to those notifications across those channels? The challenge is the literal mechanics of doing so is different in various channels, and the data that you capture is siloed in each of those providers today. SendGrid out-of-the-box, and this is true for just about every transactional mail provider, is going to track this for you. We've probably all seen it when we get an email and we hover over that link and it says, “Track.sendgrid.com.” You click that. It's tracking did the user click it?

You can instrument the other channels to do similar things, but that data is not flowing to a single system. That's one of the things that Courier provides, is you can start to understand what is that recipient profile look like across all these channels so that we can make more intelligent decisions than you otherwise could, because SendGrid might know how effective email is for you, but they don't know how effective Slack is for you and probably never can. Likewise, Twilio SMS can know to a certain degree how effectiveness SMS is for you, but they can never know how effective Facebook Messenger is as a channel for you.

[SPONSOR MESSAGE]

[00:23:39] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW CONTINUED]

[00:25:56] JM: Now, you as a routing layer that wants to be able to make decisions between these different providers, you need to be able to have access to the rate of response for Twilio, and for SendGrid, and for Facebook Messenger, and for Slack. Do those APIs – Like if you want to integrate with all of those different players, do they all expose the analytics information that you need to build Courier?

[00:26:28] TG: Yeah. The answer is sometimes yes, and in other places what we've had to do is layer our own tracking infrastructure and analytics infrastructure on top of those platforms so that we can drive that activity data into our own data stores that we can then use for, A, the intelligent routing, and B, the analytics that show you, "Hey. Okay, for this notification, these are the four channels that you're sending this notification to. Which one is performing best? As well as, For Jeff, which one is the best specifically for him as a recipient?"

[00:26:59] JM: Wow! If I integrate with Courier and I need to get data on how users are responding to a Facebook Messenger ping, you would probably have to give them – There probably have to be some cunning redirect, right? You would have to have a Courier and then have a redirect to the Facebook Messenger link, right?

[00:27:25] TG: That's correct. Yes. We'll wrap of that in a custom domain that you help provide. So we will wrap the URL that you're trying to send the user to with a tracking link that looks like your domain and then that will drive a tracking event back into Courier.

[00:27:44] JM: Ooh! I see. Right. Let's start to get into the usage. If I am using Courier and I have – Let's say it's a media company. Let's say it's the Software Engineering Daily app. We've got mobile apps and we release new podcast episodes. We release some articles, and I want to

drive engagement on those pieces of content into my app. What is my experience for integrating with Courier? Let's say I've got – When a given user signs up, I get their email address. I get their phone number and they all have the mobile apps. I've got, let's say, three delivery systems. What is my process of integrating?

[00:28:34] TG: There's two sides to it. One is the integrating it into your product, so basic calling our API. The second side is setting up the notification templates, and that's actually done inside of our user experience. That's a full drag-and-drop experience where, in theory, the engineer doesn't have to be the one to do it.

Now, maybe for you, you'd be doing both, but in large organizations it might be the product manager or the growth team member or maybe even a product marketer. They're the ones setting up the notification templates for all these channels using drag-and-drop about having to write a new code. The engineer, all they have to do is publish an event to the Courier API.

The typical walk-through of how to get started is sign up on the Courier website. It's free to get started. Free forever plan. You just click login with Google, login with GitHub. What you're then provided with is the ability to create your first notification. You create that notification, you plug in at least one channel. I don't if you use SendGrid, or MailGun or who you're using for email. Typically, people add one of those providers first, but you could start with Mobile Push or you could start with SMS. It's up to you.

You set up that first notification with one channel. You add the text. You add the call to action button, whatever else, and branding, and images that you want to have be part of that template. Now that that's there, you can kind of click a little button and it gives you a little code snippet and says, "Send this code snippet. Call this to call the Courier API." We have some libraries for – Which backend you're using. We can do PEARL, Node.JS, Ruby. It's a pure REST API, so pretty easy to access.

You then take that code snippet, you pop it into your codebase and now it'll start flying through. You got to do one other thing though, and that's either when you make that call to Courier's API to say, "Deliver this notification. You can pass Troy's contact information along with that or you

can separately send that over to Courier and we'll store that in a profile database and you only have to send that to us once.

[00:30:31] JM: Right. You have to set up each of the user's profiles within Courier because you need to associate the user ID with all of the different delivery mechanisms for that user.

[00:30:46] TG: Exactly. What we like to see our customers get to is a point where maybe they have 100 different notifications across their product that they send out. We see typically 10 to 100 is the normal range of notification count for – Let's call, it the equivalent of an early stage to mid-stage product –

[00:31:04] JM: Those are 10 to 100 different types of notifications they might send?

[00:31:07] TG: Yeah, completely different templates.

[00:31:10] JM: Right. Forgot password, if somebody friended you, somebody followed you, you've purchased something, whatever.

[00:31:17] TG: In the case of a media company, a new article is posted, a new podcast was posted, a new comment was left, and all of those account-based ones as well as you mentioned, the forgot passwords and such, billing, etc.

[00:31:31] JM: Right.

[00:31:31] TG: What we like to see is that our customers shouldn't have to go out, and when they're sending that notification to our API, we don't want them to have to go to query their user database and pull all of this information together themselves. What's Troy's SMS? What's his email? What's his Mobile Push token? We think it's easier for them to pull that once, send that over the Courier API just once. Now, whenever they send a notification, they can just say, "Deliver this notification to Troy's user ID and we'll take care of the rest."

[00:31:59] JM: Interesting. As far as sentiment about how the user is interacting with those different notification types, is there anything other than they opened it or do you try to get more

complex, like they opened it and then they engaged? They opened it then they bought something? How deeply do you get into the sentiment of the response to a notification?

[00:32:25] TG: That's a great question. Today, it's really the clickthrough tracking and saying, “Did they actually click on the call to action engage and get driven back into the experience?” That's really as far as Courier goes.

One of the things we’re working on right now is actually a segment integration, and that segment integration will be bidirectional so that you could then take that event, push it into segment, have that then route to, say, Mix Panel, or Amplitude, or Heap, or another system where you want to track engagements across your product. Then you can start to do analysis to say, “Okay, once a user has engaged in this notification, did they then go through the next four steps to say complete the purchase of their abandoned shopping cart?

[00:33:08] JM: To get into engineering, can you give me an overview of your stack and maybe some of the key technological decisions you’ve had to make?

[00:33:16] TG: Yeah. We are a JavaScript shop. We’re a Node.JS backend. I am a long time Node.JS developer. I’m on the Express JS core team, or teams about five engineers today. All of us have long backgrounds in Node.JS and React. That's our stack. We’re hosted on AWS. A fully serverless stack. Everything is running in AWS Lambda.

We've learned a lot from our past, again, in the marketing automation side of the world. One of the things that you have to be capable of is rapidly provisioning for scale at kind of in lumpy kind of timelines, right? Every early startup founder, dreams of having to support scale, right?

When you're in the notification space, you have to support scale immediately, because just one customer, they might not send that many notifications if you were to spread them out evenly overtime, but that's not the way this actually works. What will often happen is they will send of large majority or a large fraction of their notifications kind of all at once. You either have to keep a ton of idle servers just sitting there or you can leverage some of these more function as a service kind of platforms to dynamically spin up capacity as needed. There's a lot more work that goes into it than just using raw lambda, but it's an amazing start and it kind of productizes

what we had to do in the marketing automation side of the world for a long time that fundamentally boil down to just having idle servers.

[00:34:46] JM: The issue with building Courier on lambda if we were talking four or five years ago would have been is this thing reliable? Is this thing going to start up fast enough? Is this thing going to respond to a call fast enough? My understanding is that lambda has gotten much more reliable. The cold start is much less of an issue, but notification routing is really time-sensitive. You need it to be as responsive as possible. Is the cold start an issue it all for you with lambda?

[00:35:21] TG: Cold start is still – You're right, that it's improved significantly and I know they have other improvements coming soon, but it is still a thing. It's still not as good as just having idle servers standing by. There are a few things that you can do to mitigate that. One is we fundamentally have an asynchronous API. Our API is calling other APIs, and those APIs might fail, in which case we need to re-queue and retry delivery. What that means is our customers are calling the Courier API and then moving on. They don't wait for us to successfully deliver that notification because that could take a long time. Some of them might be scheduled to go out tomorrow. You don't want your server waiting for that to happen.

One, we benefit from that. Two, we put a layer in front that all it does is its job is to be as responsive as possible to your request and to queue it up, right? We make sure that there's no cold start time for you calling our API. You called API, it's always responsive, and then we queue it, and then those queued messages are picked up by lambda functions, which might have a cold start time.

The third thing is – And this is relatively recently released by AWS, is you can now set up provision to capacity with lambda. What that means is you can get back to the, "I just have idle servers sitting around world," where you can say, "Here's a minimum threshold, a floor if you will, of lambdas that I just want just warmed up sitting there." Now you're paying for those, right? You lose the hypothetical benefit of not having to pay for that capacitance.

[00:36:54] JM: So instead of scale to zero, it's like scale to two or scale to one.

[00:36:57] TG: Yeah, exactly. Exactly.

[00:36:59] JM: That's cool. Give me an idea how that scale to one or scale to two compares to like the pricing of having a single, like Fargate, instance running.

[00:37:10] TG: Yeah. I mean, it's pretty comparable when you look at the cost basically by CPU hours. Fundamentally, this is all more or less running on the same infra.

[00:37:19] JM: This is a container.

[00:37:20] TG: The pricing is pretty comparable. The nice thing is you can – Fargate is still lumpier, because you're spinning up real full servers. They're all virtual, of course, but it's not just a function. It's a real OS and you can do some really meaningful things, which is going to impact the boot times for Fargate and just some of the costs of how much memory you're going to need and how much CPU you're going to need. We've specifically built Courier so that each of the – We have really two sides to our backend. One is our delivery pipeline from when you call our API to when we've delivered that notification to our downstream providers, like Twilio or SendGrid. Then the second backend is the run the kind of design environment, and then that's a more typical web application and that's for the users that are logged into Courier and are dragging and dropping contents around on the template editor.

For the delivery pipeline, we are really, really specific and really purposeful about keeping each step of that as small as possible and a small little function as possible with as few dependencies as possible so that the spin up time, the boot time and the teardown time for each of those is minimal so we can keep the latency from when you call Courier to when we deliver it to the partner, we want the partner to be the biggest part of that in terms of time, not Courier.

[00:38:43] JM: Just going back to a said about Fargate versus lambda, I know like some of these details are like black boxed to us by AWS, but do you know more about like – You're saying that Fargate is kind of a thicker or a bigger instance when you summon it compared to lambda. I thought that in both cases, you're effectively getting a container.

[00:39:07] TG: Yes, but one is a thicker container in that. Let's say, Fargate, you are actually going to –You can do some of these with layers today in lambda. But in Fargate, it's a full OS. You can install large software pieces in it. You could install nginx on it. You can install whatever you want on it. It's the equivalent to EC2. It's dynamically provisioned EC2 instances fundamentally, whereas obviously for lambda, it's a function as a service. It's lighter weight. You have more limitations in what you can do. Yes, you could use a layer to install software. A server that's running behind-the-scenes for something like, let's say, image magic. That's a pretty typical use case, because you want to do image resizing. You can do that with layers, and sometimes that's the right choice, but you're – Really, if you're building lambda functions the right way, that image magic layer is only applied to one function, and that's not applied as a layer to the other functions in your chain, whereas you would be incentivized with Fargate to probably follow more traditional webserver rollout where that webserver has all the things on it. More of a monolith-like deployment to those Fargate instances, but you could then use Fargate to dynamically scale horizontally.

[00:40:26] JM: You said Fargate is like dynamically provisioned EC2. I thought it was containers. I mean, EC2 is –

[00:40:34] TG: It is containers. You can plug Fargate into EKS, which is Amazon's hosted version of Kubernetes. You can plug it into EC – Was it EC?

[00:40:45] JM: ECS.

[00:40:45] TG: ECS, which is their worst version of Kubernetes.

[00:40:49] JM: Proprietary.

[00:40:50] TG: Right, yeah. It is a container, but it's just like a Docker container. Think about the best practices of Docker containers. It's really to keep those as thin as possible, because the more layers of that container you have, the more things that are running in back and the longer the spin up time for that container will be.

You're right that it's all kind of – This all turtles all the way down all containers, but it's I think really the use cases that they are meant for. The way that I've seen Fargate positioned is as the successor to EC2. We think about like Kubernetes as this control plane for servers, for full servers that are essentially more or less permanently provisioned. Obviously, like one of the things it does is it will tear them down and spin them up and it needs make sure that it falls like 12 factor so that doesn't like – Don't write your data to disk and expect it to still be there tomorrow. But they're still full servers that are submits to sit there and wait for responses. It's not meant for an invented architecture where the event happens, the thing spins up, it does the thing and spins back down, which is really the function as a service lambda use case.

I think that what that leads to is while they're both similar in terms of technology, when you're using Fargate, you'd be likely to install nginx and other things on a Fargate instance and actually run a real traditional server there, whereas in lambda you are abstracting the webserver side to API Gateway and you're just receiving a JSON data structure in as your input, doing your thing and sending JSON as your output and expecting that container to then spin down.

[00:42:33] JM: Right. But from what we know about the scale to one or scale to two, whatever you call it, minimal provisioning, or what was it called for lambda?

[00:42:41] TG: I think it's re-provisioned capacity.

[00:42:43] JM: Yeah. Do we do we know anything about what the medium of runtime? Is it a container or is it like – Is it like a Fargate instance or we just don't know?

[00:42:51] TG: I don't know. The [inaudible 00:42:53] might.

[00:42:55] JM: Right.

[00:42:55] TG: I don't know.

[00:42:56] JM: Right Okay. So does that give you the degree of reliability you need to be hyper-responsive or are you saying that API Gateway is your highly reliable fronting software?

[00:43:08] TG: It's a combination of API Gateway plus the layer behind API Gateway that's responding to those events and queuing them up. Then lambda stuff takes over once things have been queued, and that's what then sees the process through to the final – Ultimate delivery to the downstream partner provider.

[00:43:25] JM: What do you use for the queuing?

[00:43:26] TG: SQS.

[00:43:26] JM: Okay. All right. It hits API Gateway, then gets queued on to SQS, and that queuing action triggers lambda, and I guess do you just have like different lambdas for the different providers? Whether it's like a lambda for Twilio and a lambda dedicated to SendGrid. How does that work?

[00:43:44] TG: Yeah, that's right. You can bind lambda functions to SQS queues. As you pop messages on to that queue, the lambda can just rip that off. When our lambda function runs, it's basically just effectively being passed in that message by the AWS infrastructure. Processes that message, potentially puts it on another queue depending on what step of the process it's in, or that's responsible for delivering to one of our integrations. Yes, we have about 20 different integrations with different partners like Slack and SendGrid and Twilio and each of those are their own lambda function pulling off of a queue, versioning it, pulling the configuration data from that customer's installation. So, what are your API keys for Twilio? Then making that outbound call to Twilio. Than we have to receive that response and deal intelligently with it. What happens if the Twilio API is down? Well, depending on the scenario, we might either re-queue that for later delivery or re-queue that as a reroute to a different channel.

[00:44:45] JM: If we take an end-to-end example, like let's say I've ordered a grocery delivery and it's coming to my apartment and the grocery delivery service uses Courier, and the system has been configured such that since the grocery delivery is on its way to my apartment, there is going to be a text message that needs to be sent to me. So give me a sense of how that message is getting rerouted, or first of all, how that triggering, how the grocery delivery service is having that triggered on their backend and that there needs to be a text message sent or there needs to be a notification sent to this person and Courier abstracts away what the

notification is going to be whether it's going to be a text message or it's going to be an in-app notification or it's going to be an email. The API of the grocery delivery service says, “Okay, notification needs to be sent to Jeff.” That notification hits your API. What happens next?

[00:45:47] TG: Yup. The grocery delivery service calls our API to pass in an event ID, and they might call that delivery notification. It might be the name of that event.

[00:45:55] JM: Right, which would be one of those 10 to 100 notification types they have.

[00:45:59] TG: Exactly. They would pass in Jeff's recipient ID. This would typically be the user ID from their authentication system. Whether they’re using AuthO or something else, they’ve got a user ID for Jeff. They pass that over to Courier. Then they pass over whatever data is relevant to this. It might be a list of the items that are being delivered, the final price and maybe an estimated delivery time. They pass that over just as a JSON data structure to Courier.

Now, Courier queues that up. So we receive that. We queue that up, and the next step that happens is a Courier lambda function picks it off the queue, pulls data about that customer, that tenant, as what we would call it internally. We look at configuration data for that tenant. We look at configuration data for that template that matches that event. We actually have a whole event mapping system, so you can start to push events to Courier without notifications existing on the other side. Then you can later come in and add a notification and say, “Whenever that event comes in the next time, I want you to send this notification to these channels.” We have to look up the mappings to figure out, “Okay. For that delivery confirmation event, which notification or notifications are going to map to it.”

We pull in the configuration and templates for that notification. That's all part of what we car our prepare stuff. We kind of bundle this up into a package that we then re-queue. The next step is what we call our route package. It calculates through looking at the user's profile data, which we actually also pulled in the prior step. Looking through the configuration, what channels have been configured for this notification? Which of those channels are still active? We have the API keys and all the other data we need. Then it kind of builds out, “Here are the possible destinations and go through the kind of prioritization and rules engine to figure out which of

those are we going to select. There are ways to configure Courier to save this into multiple. So we have to figure that out as well.

Then next step is basically queue that up for our rendering pass, which is, “Okay. We’ve decided based on this process that this notification is going to be delivered over SMS.” We basically boil down in that routing stuff to figure out, “Okay, it's going to go over SMS. So now it passes a message off on to a queue for this mess including the template that needs to be rendered and the data that needs a populate that template.”

We then have a per provider rendering engine that can – If we think about what is the format for an email notification to HTML? What is the format for an SMS notification to plaintext? What is the format for a Slack notification? Shat Slack’s proprietary JSON format called Block Kit? Microsoft teams has a proprietary JSON format, etc.

We have renderers and serializers for all of these providers. We take that same template data in, render it for that specific channel, then pass it off basically to call that downstream provider, in this case, let's say Twilio for SMS. Assuming that the Twilio API response says it was a success, then we’re done and we can mark that as a success and wait for the engagement event to kind of update that record later. If it fails, then we kind of go back a few steps and we go back to the routing process to figure out whether we want to go back to Twilio again or whether we want to go to a different channel.

[00:49:16] JM: Okay. Cool. That's a pretty detailed explanation you've given. Just to revisit the architecture, you’re fully serverless architecture. You've got to decide what data services to use. I'm curious if you've got – I'm sure stuff that you need to access really quickly. So maybe you’re using like Elastic Hash or maybe can you just walk me through the data services that you're using on the AWS side?

[00:49:43] TG: Yeah. You’ve probably gotten the impression already that we’re pretty heavily invested in the AWS infrastructure.

[00:49:48] JM: Yes. Is that accurate?

[00:49:49] TG: That is very accurate.

[00:49:51] JM: All-in or –

[00:49:52] TG: Pretty all-in. Yeah. I know a lot of companies are trying to build to be agnostic to the underlying hosting infrastructure. I think that that will be seen in a few years as the same kind of misstep we made in the 90s with ORMs where we're trying to abstract away which database layer that we were talking to.

What that generally meant is that you spent a lot of time building an inferior way to use the power and capabilities of the technology and then you never actually switched anyway. I think we'll the same thing happening with hosting infrastructure. We can take a benefit and advantage of a lot of the key capabilities of AWS. Frankly, you could for GCP or Azure if you actually just commit and build for that infrastructure.

[00:50:33] JM: Let's stable that for a second. I'll go deeper there.

[00:50:36] TG: On the persistence layer for that delivery pipeline, one of our goals is that we have basically a maximum number of queries that we're willing to call per function, because, again, we're trying to reduce the latency throughout. Every single one of those call queries needs to be a key value lookup. They can never actually do a scan. They can never do a multi-record query. We have to know exactly which key to pick up and we only do up to three calls per function to keep the latency minimal.

We use Dynamo as the persistence engine for all of those key value lookups. The data that we write into Dynamo is also streamed over to an Elasticsearch cluster. We use that in, say, our logging UI for actually querying and faceting data. But all of the kind of runtime, things that happen when you call our API between then and when we actually deliver that message to Twilio just in grid, all of that is key value lookups in Dynamo.

[00:51:32] JM: And the latency is not an issue.

[00:51:34] TG: I think we did a little demo before the discussion –

[00:51:36] JM: Yeah, that was fast.

[00:51:38] TG: And when I click on send in our little API editor, we use Insomnia, but it's like Postman. If you remember the Slack use case, the Slack beneficiation showed up before I was actually able to alt-tab over to my Slack client.

[00:51:51] JM: True. Why is it so fast? I mean, I assume if you compare like a Dynamo – This is maybe going to sound like a naïve question, but you compare DynamoDB to Redis, I would think that you would need something like Redis, some kind of in-memory data access layer in order to get the kind of latency you would need out of notification infrastructure.

[00:52:14] TG: I think as long as you're restricting yourself to just pure key value lookups, it's fundamentally hash table, and if you're just indexing directly into the exact record in the hash table, database servers are really good at doing that. That is all in some ways we are using Dynamo as effectively Redis infrastructure because we're not writing queries into the underlying JSON structures within these records. It's all key value lookups and we gain benefit – So could we use Elastic hash? Could we use the Redis infrastructure? We could, but by using Dynamo, we can do things like dynamically spin up lambda functions when certain rows are updated and we use that, for example, to then stream creation, record creation and record changes over to Elasticsearch. Again, by committing to that infrastructure, we're able to benefit and tie it into all the other services and reduce our operational complexity and our engineering costs.

[00:53:11] JM: What are you using Elasticsearch for?

[00:53:14] TG: Elasticsearch is for any queries. Anything that's not a key value lookup where we are potentially returning multiple results, those hit Elasticsearch. Those are generally only happening within the user experience when you log in as a developer or an administrator to see the list of templates and then list the notifications. Also see the logs. Every time you send a notification through Courier, we track that as a log. You can log in to Courier. You can see those logs. You can click into that log. You can see what did you send to Courier. What did Courier load for the profile? Why did Courier route it to a specific channel? What did that channel partner send back?

Let's say your call to Twilio was failing. Courier would show that to you in the dashboard. You click into those notifications and then you could see that Twilio responded back and said your API key is invalid. So maybe somebody rotated the key and forgot to update the Courier installation or something else has happened. You'd be able to see that in those logs because we capture what Twilio and what's in Grid and Slack what they're returning to.

[SPONSOR MESSAGE]

[00:54:21] JM: DigitalOcean makes infrastructure simple. I continue to use DigitalOcean because of the low friction and attention to user experience. DigitalOcean has kept the experience simple and I can spin up a server in less than a minute and get high-quality performance for a low price. For an application that needs to scale, DigitalOcean has CPU optimized droplets, memory optimized droplets, managed databases, managed Kubernetes and many more products. DigitalOcean has the flexibility to choose the right instance for the right workload and he could mix-and-match different configurations of CPU and RAM.

If you get stuck, DigitalOcean has thousands of high-quality tutorials, responsive Q&A forums and a customer team who treats customers respectfully. DigitalOcean lets developers focus on what they are building. Visit do.co/sedaily and receive $100 in credit over 60 days. That $100 can be put towards hosting or infrastructure, and that includes managed databases, a managed Kubernetes service and more.

If you want to get started with Kubernetes, DigitalOcean is a great place to go. You can use your $100 to start building your distributed system and you can get that $100 in credit for free at do.co/sedaily.

Thank you to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:55:56] JM: Are you using Amazon Elasticsearch service?

[00:56:00] TG: We are.

[00:56:01] JM: What do you think of the whole idea of like that Amazon taking open source projects and productizing them and perhaps not contributing back to the open source project and perhaps taking away business from the companies that are funding the open source project. As somebody who is advocating the all-in on AWS perspective and somebody for whom Elasticsearch service, Amazon Elasticsearch service is good enough. What are the cases where a company would need to go to Elastic and get better support? I guess we can explore each of those topics sequentially.

[00:56:41] TG: Yeah. First of all, I think that it's actually quite problematic what's happening. I would agree. I think that what we've seen are many of these companies go to the dual licensing model, and that's obviously created some controversy within the community because it's not a strict open source license anymore once you kind of go down that path.

But I think, frankly, Amazon has forced their hand. I think that either we need to go down that path or we have to abandon the idea that open source projects can generate product businesses. The Red Hat model could still work in terms of being a primarily professional services oriented business, and obviously we see a lot of open source projects go that path.

But when we look at, say, Mongo with Atlas and we look at Elastic with their own hosting infrastructure and many other open source servers are going that path, especially persistence servers, I think that they probably just don't have a choice but to go down the dual license path, because obviously they're not going to say, "Oh! Well, we should just abandon our business model.

[00:57:47] JM: So the open source project is really a considerable enough set of functionality and it's easy enough for Amazon to just spin it up and offer the basic level of UI and basic level of services on top of it such that you think it forces a company like Elastic to go with the licensing model rather than – What if they built additional services on top of Elastic or additional product offerings on top of Elastic, better search experiences, verticalized search experiences? In the case of Mongo, like we just did a show with the WebFlow former CTO, and he was saying that Mongo is so crucial to them and they really want the best Mongo experience, and Atlas

provides enough of a differentiated Mongo experience such that they're not going to go with DocumentDB. They're not going to go with the off-the-shelf thing even though they're big fans of AWS. It's just surprising to hear that perspective.

[00:58:42] TG: Well, I mean we've seen Redis go down the path of trying to add additional capabilities on top and separately license those capabilities. We saw a fair amount of pushback from the community there as well. I think there's definitely a pick your poison in terms of how are we going to get out of this bind where we want to make sure that project owners of open source projects are economic incentivized to continue working on it and the only way – Yeah, you've listed a few different options.

I think that, fundamentally, adding new capabilities on top and licensing those separate is to me not substantially different than just a dual licensing model. I think that we – In terms of who it hurts more, I think the dual licensing model hurts Amazon, hurts people who would want to host these services and be paid for it instead of the company that created the software.

I think the separate licensing for new features on top of the open source platform, which is more of the Redis model, I think that hurts more of the consumers, right? It hurts the companies that actually want to use and benefit from Redis as users, as developers, because now it's, "Okay. Well, you're kind of forcing me into do I want the free version or do I want this premium version." That feels to me less in the spirit of open source.

I do think you've illustrated one other possibility, which is maybe you're just better at hosting than Amazon, which is really an operational excellence challenge. I think it's probably a fairly separate skillset than building an amazing set of software, right?

If you can do it well and it sounds like maybe the team over at Mongo with what Atlas is doing it well enough to succeed there, then maybe that is a great model in those use cases. But I don't know that as a community we want to say you not only need to be a software engineer that is world-class at creating a differentiated persistence layer and software service, but now you also need to be world-class better than Amazon, better than Microsoft, better than Google at being able to host that and spin up capacity and provision servers, which is relatively large in business.

[01:00:53] JM: And UI layer. UI layer and developer experience.

[01:00:56] TG: Yeah. Those seem fairly orthogonal as concepts. More power to them if they can do it well, but I don't know that we want to put all of our eggs in that basket.

[01:01:05] JM: In other words, you are basically saying that the dual licensing model is the most viable path forward for a lot of these kinds of businesses.

[01:01:16] TG: Well, I'm biased. It's more aligned to my incentives as a startup founder who wants to utilize this technology. The dual licensing model either forces the Amazons, Microsofts and Googles of the world to pony-up and to actually kickback money to the creators of that software or to build their own competitors/the creator actually hosting it themselves. So we'll see what path that goes down, but it feels like a preferable path from my perspective, but I'm just a consumer of all of these.

[01:01:51] JM: What you said about the all-in on AWS perspective, I want to revisit that, because I'm with you there and I think that what you were suggesting, maybe it alludes to the fact that we perhaps have a hangover from – I don't know what the motivations were for ORMs, but I assume it had something to do with like Oracle and Microsoft and just the sense that lock-in is this pervasive problem that we can't get past. That's a hangover from a time when there was not really a check on Microsoft's power or Oracle's power and there really was not alternatives.

I mean, to some extent, if the market goes in the direction of going all-in on AWS, it would put us in a position where there's not – People who are all-in on the AWS ecosystem. There's no check there and they're totally locked in and then it's just a matter of like do we trust AWS enough not to pull the rug out from under us and increase prices a lot.

But arguably, we should trust AWS not to do that, because it's so ingrained in the culture of Amazon to consistently lower prices, just expand horizontally. That strategy has worked so well for them. Why on earth would they undermine their culture for the sake of a quick buck?

[01:03:12] TG: Still just such a competitive space and everybody is competing on price. It's relatively commoditized. I think that as long as Microsoft and Google are in the game, I don't think Amazon can do that. We've seen Microsoft Azure grow by leaps and bounds in terms of – They've clearly surpassed Google now as number two to Amazon. I think there's still plenty of competition in the market that Amazon is not incentivized to Jack prices in. It's just kind of like the historical trend line to your point is that's not the direction it goes. It goes the other way. I'm not a soothsayer. I don't know what will happen in three years. Maybe the prices will start to go up. It doesn't seem likely though, and I think that's like true of just computing in general, computing capacity, CPU capacity. Moore's law might not be in effect anymore, but it's still trending towards it gets cheaper overtime. I expect the same to be true for hosting.

[01:04:02] JM: Your point is also just accounting for the fact that there is just no free lunch. You cannot insulate yourself against platform risk. If you try to do that, you're going to fall behind the people who are not insulating themselves from platform risk and you're going to seal your fate regardless.

[01:04:19] TG: Right. It's a tradeoff, right? You're paying a cost now for a future possible savings, but it's a very real cost that you're paying now. You're going to spend more time. You're going to have a worse result today, and it may benefit you someday down the road and I'm skeptical that it would.

[01:04:36] JM: Are there any more niche AWS-based decisions or AWS products that you're using that are worth highlighting?

[01:04:45] TG: I'd say one that's relatively niche is Cognito. We use Cognito as our authentication store. All of our user accounts are stored in Cognito. We support email and password. We support GitHug OAuth. We support Google OAuth and all of that is controlled through Cognito. It's been okay. I'm not in love with it.

We do love the fact that by being all-in including at the authentication layer on AWS, all of this is automated with infrastructure as code. We can spin up entirely new production stacks from end-to-end. It takes about 10 minutes. Every developer has their own complete end-to-end stack. It's, literally, you click a button. Boom! It's up. You can spin it down. Spin it up as many times as

you want. That has been nice and there's effectively no configuration. You just click a button. Boom! There you go.

[01:05:32] JM: Is that a cloud formation template?

[01:05:33] TG: We're using serverless.com.

[01:05:35] JM: Really?

[01:05:35] TG: Yeah. It is cloud formation under the hood. But yeah.

[01:05:37] JM: So you're using the serverless framework or are you a paid customer of serverless?

[01:05:42] TG: We're not currently a paid customer of serverless. We use the open source library. I think the paid service is they have a hosted offering, which we do not use, and then they also have a support offering, which we might use at some point.

[01:05:55] JM: What else do you use outside at the AWS ecosystem?

[01:05:57] TG: Let's see. In terms of additional tools, many companies use Intercom. We have our own set of notifications that we send through Courier to downstream providers like Slack and SendGrid and Expo. We use those services, but we won't count those for now.

Let's see. We're actually about to add – We want to add some more – We have a new UI experience coming out soon and we want to be able to see the customer journey as they test that. We use Segment. We're also thinking about adding something like Fullstory. During our Y Combinator class, there was a company called Asayer. We want to check them out as well. That's kind of like a Fullstory but targeted at engineers. We're going to test out a few of those options. LogRocket I think is in the space. We'll take a look at a few of those. Datadog, we use. GitHub. Yeah.

[01:06:45] JM: Yeah. The Fullstory, LogRocket sector, those tools seems so useful. But I wonder if – I mean, it's kind of invasive. I mean, it's like you are videotaping my entire browser session.

[01:07:00] TG: Yeah, for sure.

[01:07:01] JM: Like, do I want this behavior normalized?

[01:07:03] TG: Yeah, I hear you.

[01:07:04] JM: Do you feel conflicted at all about –

[01:07:06] TG: I do, and as a callback to earlier, it's a bit of a tragedy of the commons situation where like, "Do I want to be the only one that doesn't benefit from this?" I would also – And perhaps this is just rationalizing, but I think we aren't a social media company. You're not writing really private stuff. These are templates. You're not even like writing user information in the user experience. We obviously have – We can see your templates anyway. I like to think that our customers would be more okay with us doing it than, say, Facebook. Probably they would all prefer that nobody did it.

[01:07:41] JM: All right. Well, Fullstory, a previous sponsor of Software Engineering Daily. They should still sponsor Software Engineering Daily, because there're definitely a lot of potential customers on here. But let's talk a little bit about the future. How do you see this notification market evolving over the next 5, 10 years both in terms of like new platforms and like developer tooling, developer usage?

[01:08:10] TG: Yes. I think one way to answer that is that we are seeing rapid growth of new channels, and that growth is only continuing to increase. As an example of this, a year ago, Microsoft teams didn't even make my pitch deck in terms of channels, and today it's one of the most commonly requested channels because of the growth of teams as a way to reach non-San Francisco tech companies, right? Everybody's got Office 365. So we see massive teams, MS teams growth in the Midwest and outside of Silicon Valley.

If you're a B2B company, being able to reach into that kind of always on presence that your users are sitting in all day, and they might not be sitting in your product all day is really high value. That's a channel that kind of came from nowhere and is now frequently asked for in the notification space for B2B. New channels will continue to pop up. So I didn't see that one coming. I'm sure that there will be others that I don't see yet.

We've got – WhatsApp is still trying to figure out what they're going to do here. It's really nearly impossible to get even approved for the WhatsApp business license. You can fast-track that by applying through Twilio and some other ways. But even once you do, you have to have WhatsApp approve every template that you want to send, and it's incredibly limited in what they're willing to support.

The general kind of thesis is they don't actually want to be in the notification game and they don't know whether they're going to stay there or not. So we'll see. We'll see where that goes. Basic Messenger is somewhat limited but is much further along. So we'll see some of these channels. Wechat.LINE, especially as we look outside of the US, some of these are now starting to focus on notifications.

Line, notifications for media companies is one of their primary drivers of revenue, so like the New York Times and many other media publications send their notifications in Southeast Asia over LINE. That's how they communicate with their audiences because –

[01:10:03] JM: Because nobody has the app.

[01:10:05] TG: Well, we forget that outside of the US, right? In the US, we start with emails or notification channel, because email is ubiquitous. That's not true elsewhere. You have to start to think about, "Well, what channels are ubiquitous?" and often times the answer is none. Now you're in this weird – Do you remember that carnival game where you'd have all these circles were you had to like fit the circles on top of this bigger circle so that you've perfectly overlapped that circle? You're basically like creating this Venn diagram to cover up a circle. I feel like that's what's having to happen, you have to happen with these communication channels especially outside the US where it's like I have to piecemeal all of these different channels together

because there is no one channel that I can use to reach all of my audience. We'll see a lot of growth outside of the US overtime.

We'll also receive more sophisticated orchestration of notification strategies. As an example here, I described earlier what Twitch's strategy is in terms of in-app, Mobile Push and email. Slack has almost the exact same one. About three years ago, one of their employees, Matt Haughey, posted a diagram to Slack of their exact flowchart for how they decide whether to send the notification to [inaudible 01:11:13].

[01:11:13] JM: I remember that. That's complicated.

[01:11:14] TG: Yeah. It was a funny one because Matt Haughey was like, "Hey, whenever I see somebody on Hacker News say, "I could build that in the weekends." It's like, "No, man. You couldn't even build our notifications in a week."

I've spent some time with the notifications team at Slack. There is, again, about 8 people that work full-time on notifications all day, every day.

[01:11:33] JM: Good Lord!

[01:11:34] TG: When you become a big successful tech company, you're going to have a team that's building out the internal version of Courier. They are working on some really sophisticated notification infrastructure and doing really cool things that I think are out on the bleeding edge. When we talk to other companies like Slack, companies are looking at things like doing multichannel distribution and then retracting from the channels that you didn't engage in. If I push to you on two channels and I wait for you to engage with one of them and I retract the others so it's like it never came. How can we optimize for speed of delivery and likelihood of you engaging while minimizing the impact and distraction that you might have and, frankly, your annoyance?

This is an area where I think user preferences are well aligned with what will benefit the business, because they do not want to annoy you, because you have the power to unsubscribe. It's kind of like how do I get on that knife edge of making sure you stay as informed as possible?

You keep us in mind. You kind of keep coming back into our experience, but I never cross over into that realm of annoyance where you might just cut us off.

[01:12:44] JM: Last question. What would you build if you are not building Courier?

[01:12:49] TG: Oh! That's a good question. Well, I really love building tools that make it easier for people to start businesses and start products. I think I waited too long in my career to create developer tools, and I've definitely been bitten by the bug. When I started to think about what I was going to create next, Courier was at the top of my list, and one of the reasons why is because I was like, "I've built things for marketing teams, and logistics teams, and political campaign teams. I'm an audience. I buy stuff. Why have I never built something for myself?"

I think it would be definitely something else in the developer tool space. Something that makes it easier for young engineers, young startup founders to go from the proverbial 0 to 1, how do we get that idea out into the market without having as much overhead of things I have to build that aren't core to the idea? I think Courier fits within that model. If it wasn't Courier, I'd be looking for another idea that achieve the same effect.

[01:13:47] JM: Troy, thanks for coming on the show. It's been great talking.

[01:13:49] TG: Yeah. It's been great, Jeff. Thank you.

[END OF INTERVIEW]

[01:14:00] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have Datastax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

Datastax provides Datastax enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. Datastax enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed

workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run Datastax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and Datastax's enterprise, go to datastax.com/sedaily. That's Datastax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to Datastax for being a sponsor of Software Engineering Daily. It's a great honor to have Datastax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[END]