EPISODE 1471

[INTRODUCTION]

[00:00:00] SPEAKER: Notifications have typically been an area of product were building in-house has been the only option. However, building a best in class notification system that delivers a great customer experience requires a significant investment when you start to learn complexities, like batching, cross channel orchestration, and user preferences. Chris Bell of Knock joins the show to talk through how Knock can solve the challenges with building an in-house product notification system and free up engineering time in the process.

[INTERVIEW]

[00:00:29] JM: Chris, welcome to the show.

[00:00:31] CB: Hey, Jeffrey, thanks for having me.

[00:00:33] JM: Yeah, it's great to have you. Notifications. So, I receive all kinds of notifications in my applications. I think about notifications on my phone, notifications that I see in the website of an application like on Instacart or on Facebook, and they seem fairly simple. There's a wide variety of notifications that can occur. Why is the concept of a notification a complex engineering problem?

[00:01:09] CB: Yeah, that's a great question. Just to start, I think a lot of people don't think of it as the complex engineering problem, really. But what we saw with notifications is, it's not just about the actual delivery of those notifications to the end device, right? It's not just about sending a push notification, or delivering an email or sending an SMS and calling one of those already established, like delivery provider API's, really where a lot of the complexity lies and notifications these days, is kind of just what you describe. It's about the orchestration of those notifications, and making sure that the right notification goes to the right person on the right platform to give them the highest chance of engaging with it and get them back into your product. That's where we see a lot of the actual challenges of building these notification platforms today.

So, myself and my co-founder, we worked at a company called Frame.io where notifications are basically integral to that platform, making sure that when someone left a comment on a video that they would get notified to bring them back into the product. But those notifications, they didn't just live on email, they lived in places like Slack, they lived in places like Microsoft Teams, or whatever that chat platform is that you're using, as well as like we had an iOS app, we had an Apple TV app, we had all of these different applications, and just making sure that the right notification went to the right place to make sure that person ended up getting back into the application was really, really crucial.

That's kind of the problem that we're trying to solve at Knock is not just about the delivery of those notifications, it's everything that comes around it. It's all of the logic that's involved in basically taking a flow of information and making sure that the right channel gets notified, applying a user's preferences back onto it as well. So, making sure that like, if you tell us you never want to receive a push notification, you shouldn't be receiving that, right? That's really where a lot of this engineering challenge lies and it's kind of like, it's one of those problems where it's like, it's not just a hard-technical problem. It's one of those things that's kind of laborious in the sense where everyone's building the same technology. There's no real differentiation there for most companies, and it's not necessarily something that's like making their product better. It's something that they absolutely need to make sure that they can compete and make sure that they can have a great experience for their customers, and that's really where a lot of the challenge lies, and that's exactly what we're trying to solve here at Knock.

[00:03:38] JM: Okay, got it. So, it seems like notifications are fairly domain specific, like, depending on what application I have, there's going to be a very specific set of notifications that are going to be occurring. What can you generalize about notifications in order to build notification infrastructure?

[00:04:03] CB: Yeah, that's a great question. So, what we see as the building blocks that can be generalized are these components that you basically compose to be able to create these complex notification flows. So, things like a lot of applications need batching. You might leave 10 comments on something, and it's going to produce 10 notifications by default. But you actually want to say, "Hey, we want to collapse those into a single notification that goes out within a set window of time." That's an example of one of the kind of like generalizable building blocks that we saw that we created around with Knock so that people can leverage those to not have to build that themselves.

Another example there would be like a delay, right? You might have an example of an invite email where you basically say, “Hey, Jeffrey, you've been invited to this product. We're going to wait three days and if you have not clicked on that invite email or accepted it, we want to send you a reminder.” It's like those kinds of building blocks that we're trying to abstract, and they're the parts that we see as incredibly generalizable, that a lot of teams are having to build themselves with – the one I just described there around like a delay is generally been built as like a cron job, but now you have to keep stating your DB. You're thinking about basically like, okay, for all these users that haven't received a reminder, we're going to do that. Again, it's just like, one of those wrote tasks that you're adding to your engineering team's backlog, and they're the kinds of pieces that we're trying to abstract, and make really, really generalizable for people to leverage here.

Another really good example is that application of preferences as well. So typically, that's not really domain specific. A lot of folks will just have like a generalized preference model that says, I want to receive emails, I don't want to receive push notifications. Maybe for this particular type of notification, I always want to receive it on this channel only, but I never want to receive it over Slack or something like that. Again, and that's one of those pieces that we've kind of abstracted away and included in the product. So, you just never have to think about building it again.

[00:06:11] JM: Okay, actually, I have built multiple products where I've had to build ad hoc notification infrastructure, and it actually is kind of a nightmare. Now, as far as what kinds of problems you can actually handle for the developer, there was this problem that Facebook had, maybe you remember this many years ago, where the user’s notifications would never show that they were empty. They never seem to be able to successfully mark all the notifications is being read. So, every time you would open Facebook, you would have a notification and it just turned out that there was some kind of complexity in terms of actually identifying what notifications had been consumed or not. I think, it was some kind of issue, communicating between the back end and the front end. But it just makes me wonder like, what parts of the notification infrastructure are you trying to handle? Is it mainly the back-end side of things? Are you also trying to handle like the front-end rendering side of things?

[00:07:15] CB: Yeah, that's a great question. By the way, we've definitely heard about that problem from some other folks as well. I think like, when you think about your Twitter DMs, they also have a similar problem, like, your badge count on iOS will never reflect the actual DM count. That kind of state

synchronization problem is definitely difficult challenge across clients, and at scale, at the kind of Twitter, Facebook scale, I'm imagining that's like a lot more challenging to handle as well.

To answer your question, yeah, we are basically, we're trying to tackle notifications holistically. So, one part that we see as really integral to that kind of holistic solving here is actually bringing in that in app kind of stateful feed component as well. If you think about feed of notifications, you might call that an inbox or you might refer to that as you know – you might have seen it as like the bell icon where you click it and then you see a list of unread notifications, something like that. We actually have out of the box React components that you can drop into your product, and get up and running with a feed that we've already constructed for you, that you can just drop in, and it gives you kind of a real time enabled feed that you can just drop into your product, out of the box, and you have that. But we also have the API's behind it.

So, if you don't want to use our components, you can also kind of power your own components through our API's and not have to build the API's to actually power that, and all of the real time infrastructure that surrounds it as well. And then, on the kind of client state management side of the house, we give you a really simple interface where you basically have access to the total number of unread notifications, things like that. You can mark them as red, just like, basically, again, just taking away that kind of task of having to build out that part of your notification stack and doing in your product, whilst also understanding that a lot of the time those components have to be really customizable, because you want it to match your branding, you want it to match the look and feel of your product. But again, it's just like rote work that you have to do and maybe you don't have real time infrastructure ready to go just out of the box. You have to think about how do you handle that. It also just brings in all of these other challenges. So, we're just trying to make it absolutely straightforward to get going and power both your back end and your front end of your notifications.

[00:09:32] JM: So, maybe we can characterize a few different kinds of applications and talk about what it would take to build notification infrastructure for these different applications. So, maybe we could start with a consumer application. If you think about like an app like, oh, let's say Instacart or DoorDash, some kind of food delivery app, can you talk through – if I was building notification infrastructure for a food delivery application, could you contrast what that would look like if I was building it from scratch versus what it would look like if I was building it with your infrastructure at Knock?

[00:10:21] CB: Yeah, absolutely. So, I think if you're building that kind of infrastructure for a consumer base application, like Instacart or one of those food delivery services, basically, what you're having to do there is think about, well, let's think about the key events that are causing notifications, right? So, you've got pretty much when an order is placed, you probably want to send them a notification telling them the order is placed, and maybe that's a receipt, or some kind of email or something like that, then you want to tell them the driver is on the way, you want to keep them updated on that part, maybe it's something to do with the fact that they couldn't find all of your groceries. So, now we have to come back and actually start thinking about bringing them back into the product to make sure that you're picking substitutes or something as well. So, you've got that kind of engagement loop that you need to power.

And then ultimately, you've got the kind of the real time nature of making sure that they're aware that the driver is coming up to their door and nearby. You've got those kinds of events and this is how we like to start thinking about kind of notifications within products. There are key events that are driving those notifications. If we're building this out ourselves, well, first of all, we just identified a couple of different platforms where we need to start thinking about sending notifications to there, right? Maybe let's start with the fact that we need to at least send them emails, but we also know that they have a mobile app, and now that we need to start sending push notifications as well.

Immediately, we have a couple of channels that we need to support. So, I don't know last time you spun up any kind of push notification infrastructure, but that's a stateful connection to APNS or FCM, Firebase Cloud Messaging as well. Now, we need to support that within our application, make sure that we're actually delivering those notifications to those end users can spin up the infrastructure to handle that. Now, on the other side, we might have to have some kind of event pipeline to basically start saying, when this event happens, we want to asynchronously deliver these notifications. Maybe that starts out as a simple job queue to actually do that. But then quickly, it can turn into like an event log service, like a Kafka or a Kinesis, or something like that as well to start powering a larger scale. And then we might have some more reminders for those users, maybe like 10 minutes away from when the delivery supposed to happen, we need to trigger a workflow that says, “We need to send them a reminder that their delivery is coming.”

So now, we might need some cron infrastructure as well or something so that we can run those kinds of scheduled tasks in the background. Now, all of those things might sound quite straightforward when

you're getting started. But again, when you're building your startup, you don't necessarily want to be focused on this part, like this notification infrastructure side, I think, really, you're wanting to be focused on like differentiating parts of that product, which is really the customer facing like UX and UI around these kind of delivery services. That's the part that you should be focusing on as a team and not having to split up all of these extra pieces of infrastructure to handle notifications. If you contrast that with what you can do at Knock, well, you can start plugging in Knock really, really easily. Basically, we take your events that are happening in your product. You send us over an API call that map's that event into what we call a workflow with a Knock, and that workflow can trigger notifications that end up at different places. Again, that might be your email, that might be push notifications.

But then let's add in another dimension where let's say you want to start sending customer's SMS for some of these really urgent notifications as well. Well, if you're doing that yourself, that means you're going to need basically another delivery service. Maybe that's Twilio, and now you're actually calling another API in some of that notifications code that you've written. Whereas in Knock, you're basically just adding another step to these workflows. You give us your Twilio credentials, we're going to manage all that delivery on your behalf, we're going to make sure that those messages get reliably delivered to your end customers as well. And again, it's no extra work for your team. We've just handled all of that part for you. Those reminder messages. There's no cron infrastructure, there's no push infrastructure to set up. Everything is just taken care of.

[00:14:38] JM: You've described a lot of back-end infrastructure there that you're using to support your notification infrastructure. Can we talk through some of the infrastructure decisions that you've made to support that back end? Can you just talk about your basic approach to what you're using for compute and caching and storage and just give me the rundown of your infrastructure?

[00:15:05] CB: Yeah, absolutely. So, we broadly separate out our system into two pieces. One is a control plane, where you are kind of managing all of these notifications that you're creating. And the other is a data plane, which is effectively the execution engine for our notification kind of infrastructure. So, that kind of stateful workflow engine that we have.

Broadly, the way that we deploy right now, we deploy on EKS, everything's in Kubernetes. We use Kinesis upfront to make sure that we can handle busy traffic, which is really, really important for us. And then all of our back-end services are written in Elixir. Elixir is a functional language that runs on the

Erlang virtual machine, if you're not familiar with it. And that's where we actually execute all of our notifications. We have pretty good amount of caching that we introduce for some of our feeds, and some of the more kind of stateful red components here, where we're using Redis for that right now. And then all of our kind of database layer, we use a combination of both MongoDB for lots of unstructured data, and then we use in a lot of Postgres, as well, for a lot more of this kind of relational data that's been stored here.

Our infrastructure is set up so that we can horizontally scale really easily across these clients. One thing that's really, really important for us is, you might be a small startup sending like 100 notifications a day, or you might be a huge company sending us thousands and thousands of notifications a second, ae need to be able to basically support that at scale. And then a lot of the rest of the infrastructure is really leveraging a lot of the kind of niceties of Elixir and the Erlang VM as well. So, the way that we actually execute on notifications is, every notification is modeled as a process in our system. Processes in Elixir are basically, they don't map to a thread. They're very lightweight, VM managed processes. They're extremely lightweight, and in fact, you can run millions and millions of them on a single like MacBook. All of our notifications are modeled like that and they're stateful processes that can be started and stopped.

So, if you imagine it more like an actor model of how these things are modeled, that's kind of maps one to one with how we think about notifications. We think that Elixir is a fantastic fit for building what we're building. Erlang was kind of developed off of telephony systems in the ‘80s and ‘90s, and I think when you think about like, kind of notifications, we actually call our back-end service for sending notifications switchboard. So, it's a very good analog for what Erlang was developed for and these kinds of processes that can terminate really gracefully, they can handle their own state, they can be retried. And we have all of these nice attributes based in the system, and this really good concurrency model just because we're leveraging Erlang and Elixir out of the gate.

[00:17:58] JM: The usage of Elixir, I guess, I'd like to go a little bit further into that. I can kind of imagine the usefulness of that, because it's kind of a routing problem. Every time that a notification gets issued to your back-end infrastructure from whatever company is plugging into your infrastructure, it has to get routed to the right client, or clients. And Elixir, Erlang are generally used – the cases I've seen them used as in telephony type applications, like WhatsApp, or historically in other telephony, like, obviously, it came from Ericsson, historically. And then also just in Pub/Sub contexts where you need low latency,

highly reliable Pub/Sub systems. So, I can imagine you have a notification and that notification may need to be sent to multiple clients, or routed to multiple clients of the same user or different users, and I guess maybe you could give a description of why Erlang is a particularly good language choice for this application.

[00:19:22] CB: Yeah, absolutely. I would say one thing about the Erlang VM, which is called the BEAM, the virtual machine underneath is basically going to manage the kind of the execution of those processes against the underlying resources of the system. And I would say, that allows us out of the gate to basically have a really simple concurrency model where again, we can kind of think about the fan out of these notifications to – it might be one request comes in and needs to end up notifying like a hundred different users or maybe it's higher than that, any kind of – each one of those users and each notification that's been executed there, we represent as its own processor on the Erlang VM. Basically, what that allows us to do is get very high concurrency, allow us to safely kind of execute all of these processes where they can fail independently. But that's not going to crash the system, those things can fail, and not actually have any larger repercussions based on the supervision model that Erlang and the system underneath is called OTP, which actually gives us really nice guide rails and guarantees in terms of making sure that we are isolating failures, we're making sure that the system can keep running a very high load because of the virtual machine and how it's managing, switching and priority between different processes as well.

There is a fantastic talk called the Soul of Elixir and Erlang by Saša Jurić, that was at the GOTO Conference. So, it's just like, probably gives the best example of like, why you might want to consider the use of Elixir in your systems. I've actually been an Elixir engineer for probably six or seven years at this point, and I actually don't think there's been a better application for what I've been writing than the Knock in terms of like the use of Elixir here, and the fact that we do have this very high concurrency kind of kneading system, and we get the benefits of that through the Erlang VM plus all of these other niceties. Plus, we get this like really nice way to express our logic, and I would say, ultimately, that is the other part here. Yeah, we have the semantics of the actual underlying VM and what's happening. But on top of that, we also get this really nice programming language that makes it a real joy to think about kind of data coming in the system, the transformations of that, and then the side effects of that being the delivery to these end providers and such.

But we find it to be a really lovely way to kind of express this logic and execute and write our workflow engine, which ultimately is really the heart of what we're doing at Knock, just because you can express all of these different ways of thinking about like an event leads to these different steps that gets executed. So, one of those steps might be a batch, one of them might be, “Hey, we need to delay this notification for some amount of time, and then we need to pick it back up and send out an email.” But we don't want to send that email if someone has their preferences turned off. There are just all of these kinds of problems that have folded inside of this, that we feel like writing it, and Elixir gives us a really expressive way of writing this kind of state engine effectively.

[00:22:28] JM: Can you walk me through the lifecycle of a notification? Like let's say, I'm on that food delivery app, and a notification is being sent from the food delivery company that my order is being picked up from the restaurant. Walk me through the lifecycle of that notification and where it hits your infrastructure? Where it's being stored, et cetera?

[00:22:57] CB: Yeah, sure. Knock is an event driven system, again. So, everything that happens here, you basically model off of that event being triggered in your system. Let's say that the food is picked up, let's just pretend that we know how that that happens in whatever food delivery service that we've already built, and then what you would say at that point in time is, you'd basically make an API call to Knock that says, “Hey, Knock, we want to say that this food has been picked up, we have a notification workflow that's associated with that event”, and then we tell Knock about who needs to be notified for that event happening. So, who is the actor of the event? Who performed it, and then who are the recipients of that event? And then we might want to pass some arbitrary data across as well, that helps us in our notifications that have been sent out.

Maybe in this example, it's like an order number or some information about that order, or who the driver is or something like that. So, that's what we do there. We make an API call to Knock, then that API call is going to hit our infrastructure, we validate, and make sure we haven't seen that before. We make sure that your API key is correct, and such and things like that. We're going to queue that in Kinesis, to make sure that we're not going to be overloaded on our site, and we basically want to respond to you as quickly as possible to make sure that getting a low latency there. As soon as we can put it into Kinesis, it basically gives us guarantees around retries, and making sure that we can reprocess those events if something happens down the line.

And then basically, what's going to happen inside of our infrastructure is that notification that you invoked there, that is going to have what we call a workflow associated with it. That workflow is basically going to get loaded from our system. We keep those in memory in a cache, and then we're basically going to per recipient that you've told us to notify, we're going to execute that workflow for each recipient. What that means for us is executing our workflow engine, going through all of the steps, you might need to hydrate some extra data in there, because notifications end up being pretty stateful, especially when you start thinking about things like batching and if we were going to collapse those notifications together, we might need to know about previous attempts that you've tried to do within some timeframe to say like, “Hey, these are all connected together”, and then we're going to execute it.

Ultimately, if there are what we call messages, which ends up being the actual notifications that you receive on your devices that need to get executed there, we are going to load the templates that you've already preprogrammed into Knock around those notifications, because what we actually try to take care of and Knock is like, there's two sides of this problem. There's the infrastructure side, which is just like purely executing your notifications, and then there's the kind of like, management side of all of these notifications as well. So, the templates that are associated with them that a lot of the time, your engineers are having to manage that in their back-end code base. But we actually hoist that out of your back-end code base, we keep it in Knock, we allow you to version control those things, we have a git like model for making sure that everything's version controlled. Everything's isolated between environments, so that you can basically make sure that things that you're doing in your development environment don't affect your staging environment, so you can really confidently make changes to your notifications.

Again, it's kind of, the goal here is to take work away from engineers who are a lot of the time often having to manage those notification templates. And ultimately, that's work that we believe is kind of wasted for engineers to be doing and should be owned by other members of the team, like product managers, or maybe its growth marketers or something like that.

So again, going back to the execution of those, we’ll then fetch all those templates, we’ll execute the templates on your behalf with the data that you passed in, and then we'll then queue messages for delivery to those end providers that you've configured for us. Let's say that you're using SendGrid for your emails, so we might then effectively generate an email that's going to go to that end provider to SendGrid, that tells you about that, that your food is on the way. And then maybe you also set your

preference to say, “Hey, I also want to receive SMS as well.” So, we're going to make sure, see if that recipient has a phone number associated with it. If they have a phone number, and they've opted in to receive SMSs, then we're going to actually call and queue another message that's going to go to Twilio to their SMS delivery product as well.

And then on behind the scenes, we're storing a lot of state here, we're keeping all of the logs that are generated along the way, so you have fantastic introspection into what's happening. In Knock itself, you can literally see from your API log that's come in, so that actual API call that you made all the way through per recipient and see every single step on the workflow that's executed. So, Jeffrey, let's say that you opted out of receiving SMSs, we'll show you a diagram, and we'll show you a logline that says, “Hey, Jeffrey never received an SMS because he opted out of receiving them.” And we keep all of that logging information for you and we give you this really, really easy way to introspect the system, and these kinds of workflows that have been executed. And ultimately, that notifications that have been generated at the other end, as well.

We'll also then keep all of the logs that are being executed against those delivery providers. So, you can see the exact call that we were making to that delivery provider, you can see if that call failed, you can take action on it if it did fail as well, and that's kind of what we're keeping behind the scenes. A lot of that kind of state is saved in both databases, back into Kinesis queues, logs out in S3 and various places as well.

[00:28:42] JM: All the layers of reliability and failover, it seems like there's some redundancy of storage and where you're putting the notifications. Now notification is not that big, but I imagine if you're handling a lot of them, then it could add up. Do you have like a garbage collection mechanism where you wipe out old notifications?

[00:29:07] CB: Yeah, we do. So, we've been relying pretty heavily on partition Postgres tables for the time being, and then that gives us a really easy way to just kind of drop old tables and make sure that we can purge those messages. And then we're looking to basically back those up into S3, so that we have a permanent store of it if we ever need to it, especially for those more kind of privacy aware customers in the kind of banking or maybe it's healthcare kind of space. So, making sure they have access to that data for over a year or so if they need it. But yeah, we keep everything for at least a month in what we call like a hot storage in a database, and they're in partition tables.

[00:29:46] JM: Got it. Alright, so we talked about kind of a consumer application. There are other applications where the notification load could be much more severe, like a monitoring application, for example, like an infrastructure, something related to monitoring your infrastructure. If there's a high volume of notifications per user, does that increase on load? Does that lead to any added complexities for the notification infrastructure?

[00:30:19] CB: For us, yeah, sure. But we think of that as a job of like something that we're providing for you and we see that as a core competency of our service, like, we need to be able to handle a great deal of load, that's just part and parcel of operating a piece of infrastructure like this. So, it's definitely part of the kind of responsibility of me and my team here to make sure that we can scale up that infrastructure, we can support really high volumes of messages over time, and make sure that whether you're an infrastructure company sending us millions of events, or whether you're a startup sending us hundreds of events that turn into notifications, we can execute that. What that means for us is putting some customers on kind of dedicated shards, and making sure that we can scale them differently, depending on the load.

We run a classic kind of multi-tenanted SaaS app, where we have the ability to do that. And again, it's like one of those things where we have to provide that service and we have to be up. If you're going to depend on us, we know that notifications can be something that's on the revenue path for a lot of these businesses, right? They can be really critical. You can stop sending notifications, and you can see your engagement drop, or you can stop sending notifications, and you can see your NPS score dropped, because people like I never received them, or I receive too many, and they can be a real liability on the business. We know that and we want to make sure that if you're using Knock, you trust us, and that we have a fantastic amount of uptime to support whatever load it is that you are calling us with and making sure that we can do that reliably at the scale we're doing today, and a scale way into the future as well.

We've invested a lot of time and energy, so far, in load testing our service, making sure that we have a lot of headroom to make sure that we can scale up and down and really elastically in terms of taking in this kind of, it can be incredibly bursty traffic, because ultimately, we don't know what you are going to call us with as the end customer. Again, in that kind of infrastructure example that you gave, we actually, we have a couple of customers who are using us for that or looking to use us for that as well. They could be sending us like thousands of events and a lot of those, we need to basically drop

because maybe they're dupes or maybe like you got throttled in terms of like – Jeffrey, maybe you say, I only ever want to receive one alert per hour or something like that, we would handle that logic for you and make sure that we're dropping any events that shouldn't lead to notifications if it's within that window of time.

So again, really, from the get go for us, ever since we started this company, I think about it as having two real big problems to solve. One of which is the kind of management side of these notifications and the other is just, yeah, making sure we're building like highly reliable, scalable infrastructure to support the load that can be coming and that we obviously as a young startup, we hope comes and that we're already seeing today, and that we hope just keeps growing and growing into the future and making sure that our servers can support that at whatever scale that is of the businesses that we're taking on.

[00:33:30] JM: When you look at the opportunities in where to go from notification infrastructure, what are the additional product verticals you expect yourself to be able to go into?

[00:33:48] CB: Yeah, it's a great question. When we started out building Knock, we did a lot of customer exploration. We talked to a lot of people along the way, and I think one thing that was really, really interesting is, you hear some teams that are leveraging these kind of more traditional marketing tools in order to power their product notifications. But one thing that we think a lot about is the kind of unification of customer messaging. So, what does that mean? Well, I think in most SaaS businesses, and most startups and maybe a lot of consumer companies as well, you kind of have this divide between the marketing team who are sending out various messages to your end customers. Maybe that's about growth marketing, and getting people in and sending them promotions, things like that. And then you have your kind of product centric notifications that are going out to notify people about the events that are happening, and more of the transactional notifications.

But one thing that we keep thinking a lot about is, I think you see this in some of the like best in class companies out there where they're actually kind of thinking about that holistically, where it's not a separation of messaging, it's just a single bucket of messages that are getting delivered to a customer with basically a prioritization of those messages to make sure that you're not sending too many. If someone starts opening your emails, maybe start sending them more. So, we really think about the pushing into this kind of broader space of customer engagement and customer messaging, as being a real opportunity for us. And coming at it from a different perspective than the marketing teams

necessarily, and all those marketing products like Braze, Leanplum, and folks like that, and more coming at it from this perspective of unified customer messaging that your product team is owning, and you're really thinking about the best interest of your customers, making sure you're not sending them too many messages, making sure you're approaching them on the right channels, making sure you're kind of delivering these messages at the right time, so that you can get higher engagement and ultimately deliver higher customer satisfaction as well.

[00:35:53] JM: If you look at developer tooling, as a category, it's a very good category to be building in because users rarely churn and the spend of a user just grows over time. Do you have any commentary on selling to developers?

[00:36:15] CB: Yeah, I have lots, yeah. I think your points are really well taken there. Like we love being a developer tool. It was one of the reasons, as well as the problems that we saw that we wanted to build this business and that I feel like building usage-based products is really awesome. Your customers succeed, and you succeed. That's fantastic, right? I think selling to developers is a really, really interesting problem. I'm an engineer, I've been an engineer. I'm an engineering leader now and I think about the tools that I buy, and the things that I think are really great out there, and that's exactly what we're trying to do with Knock. I talk about this as like, you can have a fantastic product, but if it's developer focus, unless it passes that like smell test for engineers, they're never going to adopt it, as a product.

I think about the companies who are kind of like best in class here in terms of excellent developer experience, and that's the kind of bar that we're looking to hit and what we're trying to do here. So, what does that mean? Well, for me, that's like, I think there's the obvious ones, like fantastic documentation. I think you've seen like people like work OS, you've got your stripes, people like that who just do a really good job of the writing that's around their documentation, and also the guides, and also the level of effort that's actually involved there. Making sure that the SDK examples are excellent. It's really easy to toggle. Then you end up with all these like really nice details where maybe your actual API key is pre-populated in those examples to make it really easy to run it, things like that. And then I think about having fantastic tooling on top, I think, the first example that comes to mind to me for this is like planet scale with their CLI tool, just like so, so good and really hits the nail on the head of what engineers want and how they want to interact with the service and what they need to get their jobs done.

And then I think the other pillar here for me is like, fantastic SDKs, as well. That means like well tested, really well documented SDKs that make it really easy to contribute to, if you want to, but they're very easy to get going out of the box, they have like a very sane API design that makes it really straightforward to get going as well. Again, I think the canonical example in this category is obviously Stripe. We wall continuously point to Stripe and I probably have done for like the last decade, and I will keep doing that because they do such a good job in terms of how they think about building for developers and the products that they're creating. I think at Knock, we're really trying to embody a lot of those ideas and really make it part of our DNA from the get go, where we're kind of poring over the API design that we're introducing. We're making sure like we're being really thorough and thinking about consistency across our API endpoints and documentation and just all of these parts of the experience because ultimately, yeah, we're selling to those engineers, and they need to trust us. We want them to feel like it's a delightful product to use in terms of the problem that it's solving. I think if you read some of like, the commentary that we got from some of our existing customers on our recent product release, I feel like we had a lot of love for the way the API's are designed, the way the SDKs are so easy to use as well.

I feel like we're on the way there, but it's this is whole part of the challenge of building a business in this kind of category. Just making sure that you are absolutely crushing the developer experience and thinking about it day in, day out across the team, no matter who it is on the team and what they're doing.

[00:39:59] JM: We've talked to some engineering decisions. Can you give any examples of some very difficult engineering decisions that you've had to make that we haven't talked about?

[00:40:09] CB: Yeah, absolutely. I think like the difficult ones for us, I think a lot about building an early stage infrastructure company as like one of the biggest difficult decisions that we made. The part that comes to mind for me around that has just been the tradeoff between time to market and reliability and scalability, right? In terms of, you're building a company, you're building a new business, you don't necessarily know how far it's going to go. You don't know what stage you're going to get to. But also, when you're building an infrastructure company, you don't want to like half asset it, so to say. You want to make sure that you're building on a really good foundation that can scale and that you are kind of taking into account, what's going to happen when you get those huge customers? How are you going to shard? How are you going to manage this platform?

So, I think that tension, and that tradeoff between time to market and scale has been probably the thing that we have not labored over, but just like, has been one of those really hard decisions about what do you do now? Versus what's the tradeoff for that decision for where you're going to go and when you're going to end up, right? We try to capture a lot of those early decisions in a decision log, where we know we're making a tradeoff for something that we've adopted now versus something that we know that we're going to need to swap out down the road. And then also, understanding what the headroom is for how far that's going to take you before you know you need to swap that out. And then prioritizing in terms of just like, we can get to this point on this technology, but we know beyond that, we're going to need to make a different decision, or we're going to need to shard that in a different way or whatever it might be, and knowing when you need to prioritize that and keeping track of those has been a really big thing for us that we've really tried to adopt early as part of the culture. Again, we kind of did this riff off of the decision log idea and keeping track of all of those things. So, everyone on the team knows why we made those tradeoffs and knows why we made those decisions and knows where the kind of the headroom and at the end of that scale journey is for those pieces and tools that we've made.

I know that's slightly vague. But I would say that that has been the biggest thing that we've kind of wrestled with so far in our journey to get to where we are, and be in market with sending a ton of volume and notifications.

[00:42:31] JM: That's interesting. It's kind of abstract. What about, do you have any anything like closer to the metal that was difficult?

[00:42:38] CB: I have loads of stuff to say about this, honestly. As a young company, that was like spinning something up, I think people have railed on the idea of using Kubernetes as an early stage startup a lot. But the amount of ops and infrastructure work that comes off of that decision is actually pretty phenomenal, in terms of – we adopted K8s because we wanted a good foundation that we know we can horizontally scale, and we know we've had experience as a team operating that in the past. So, we didn't feel too bad about like using it again here.

But just that zero to one experience of, we wanted to Terraform all of our infrastructure from the get go so that we had a really good infrastructure as code story for when we went out and did our security work. And then we knew that now we need a way to, maybe we need to access one of those boxes, but

everything's in our VPC. So, we need a way to like SSO in we want to set up everything on AWS with best practices out of the gate, the learning curve for Terraform ops, and Infra is so steep. It's so much work that we had to do to just get something that was good, working, and allowed us to deploy. I think the investment there for us was really worth it, because it allowed us to very, very easily go out and get our kind of SOC 2 certification based on how our infrastructure was set up for security best practices out of the gate.

But just the level of effort, and the amount of time that it took just to have something that was working, and that we could build on top of was an enormous, and I think a lot about like, as an early stage company, again, that isn't your priority. But when you're an infrastructure company, it kind of is. You want to make sure that what you're running on is good, it's easy for you to scale, it's well provisioned, it's really secure. So, you've got all of these things to think about, and I don't know, I'm not the biggest fan of doing that kind of work, especially around like Terraform and ops. I am really thankful that we had some great contributors in there as well, because I was also like, trying to be heads down building the application at the same time. But yeah, it was definitely a challenge in terms of just getting to market with that and basically setting ourselves up for the future and the level of investment there, for sure.

[00:45:00] JM: Did you use like a managed Kubernetes system like AWS EKS or something?

[00:45:07] CB: Yeah, we use EKS to do – but that takes away some of the management part. But there's still a lot of work in terms of VPCs, in terms of just making sure everything's provisioned correctly, making sure that you've got good security best practices out the gate. IAM is a whole pain to actually be thinking about like, least privilege, making sure you're doing all of that. So, we used a lot of managed services along the way, because we're a really small team, and we don't want to be running databases and running clusters too much, but there is still a degree of stuff that we had to do there. I don't think that this necessarily goes away entirely with anything you're doing with serverless, as well. There are still challenges and making sure that you can kind of think about spinning up and deploying, and all of the rest of it, the CI/CD processing just getting everything down. So that, as an early stage team, velocity is the thing that you should be aiming for. Your ability to ship product is like, it's the heartbeat of the company and anything that slows that down for engineers is a bad thing. So, we spent a lot of time on that kind of CI/CD story, making sure our tooling was really good, invested in that early, so we can get to market quickly and keep iterating as quickly as possible.

[00:46:31] JM: Any closing thoughts? We've talked through a lot of different elements of Knock, what else would you like to share?

[00:46:39] CB: I would love to say that, I think, this notification problem, if you're experiencing it today, and you're looking at like a build versus buy, definitely, I think there's a lot of companies out there who will not treat this as a build versus buy decision right now, and will absolutely just go ahead and build something inhouse. I want to say as someone who has run a team that has had to manage this, and has had to build that system inhouse, it starts off easy, it's a ball of mud that you're going to just keep slapping things on to, and we really believe that you should have a little look before you make that decision at some of the tools out there in the space. We think that Knock can be a really compelling offering just in terms of you not needing to build a notification system, giving you fantastic abstractions so that you can just make API calls, and that's the end of your kind of journey of thinking about notifications. So, definitely have a look there.

The other thing I'd love to plug as well, just while I'm here. I do a lot in the Elixir community. I think it's a fantastic community that's still fairly under the radar, fairly underrated and the power that it kind of brings to the table. I think Elixir is a really great tool for writing companies like this and for writing systems that need to be reliable, scalable, and just really lovely to maintain as well. So, there's a conference coming up called EMPEX MTN that I am hosting. It's May 6th in Salt Lake City. We'd love to see folks there. If you're thinking about Elixir, or like you're interested in it, we've got a fantastic lineup of speakers. So, we'd love to see more folks from other communities coming and dipping their toes into the Elixir world and playing around with the language and hope to see more people there as well. Thanks.

[00:48:29] JM: Awesome. Chris, thanks so much for coming on the show.

[00:48:31] CB: Thank you very much, Jeffrey. It’s a pleasure to be here and really great conversations. So, thank you.

[END]