EPISODE 554

[INTRODUCTION]

[0:00:00.3] JM: The pub-sub pattern allows the developer to create channels and messages can be written to and read from those channels. Pub-sub messaging is useful for multicast messaging when you want to publish messages from a producer and have multiple consumers who are subscribed to the publisher receive those messages. Almost any application that reaches a high level of complexity will need a pub-sub system of some kind, and even simple systems like chat applications are made much easier by a pub-sub component. A pub-sub system itself can be complex. A pub-sub system needs to scale up and down to handle different numbers of consumers and producers in different volumes of messages.

Back in 2010, the growth of mobile and cloud was leading to many new applications with high throughput multiuser interactions. Developers were standing up their own instances of open source pub-sub message queuing systems like a RabbitMQ and ZeroMQ. Once the MQ systems needed to scale, the developer would need to handle the scaling themselves.

Stephen Blum started his company, PubNub around this time to create automatically scaling APIs for messaging. Stephen joins the show to discuss the infrastructure choices around building a large scale pub-sub service and how the company has scaled over time. He also talks about the management, product development and business side of running the company. PubNub has built several additional technologies on top of the core infrastructure that was originally built for pub-sub messaging.

Full disclosure; PubNub is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:49.4] JM: LiveRamp is one of the fastest growing companies in data connectivity in the Bay Area and they’re looking for senior level talent to join their team. LiveRamp helps the world’s largest brands activate their data to improve customer interactions on any channel or device. The infrastructure is at a tremendous scale. A 500 billion node identity graph generated

from over a thousand data sources running a 85 petabyte Hadoop cluster and application servers that process over 20 billion HTTP requests per day.

The LiveRamp team thrives on mind-bending technical challenges. LiveRamp members value entrepreneurship, humility and constant personal growth. If this sounds like a fit for you, check out softwareengineeringdaily.com/liveramp. That's softwareengineeringdaily.com/liveramp.

Thanks to LiveRamp for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:02:56.4] JM: Stephen Blum is the founder and CTO of PubNub. Stephen, welcome to Software Engineering Daily.

[0:03:01.2] SB: Hey! Thank you for having me today.

[0:03:02.7] JM: Yeah, it's great to have you. You founded PubNub which is a play-off of the term pub-sub, and the pub-sub pattern has been around in software engineering for a pretty long time. Explain with the pub-sub pattern is.

[0:03:16.3] SB: Yeah, great question. It's funny that you even asked that in the first place talking about PubNub being a play on the word pub-sub. Specifically, we did that for marketing reasons. If you search for PubNub, you're only going to find PubNub. If you search for pub-sub, you'll see a whole bunch of stuff. Publish-subscribe is that paradigm where you have a centralized topic or a conversation, sort of like chat room. You can say I want to subscribed this chat room and anything that happens in that chat room I want to be notified immediately. You can publish information to that chat room and everyone subscribed to that chat room will get that update. It's like IRC, Slack, you're familiar with those types of applications that use publish-subscribe.

[0:03:52.5] JM: Although it's not just useful for chat systems. This is something that's widely used for — I mean, it's used for all kinds of applications. Generally, when you scale and you have types of systems where you have multiple consumers off of a single producer, for

example, and you need to do kind of specific casting, where maybe you're not multicasting to every single channel that's out there that people could potentially listen to, but you want to hit a specific subset of applications that are listening. The pub-sub patterns is quite useful for that.

So when you started PubNub, there were few ways of doing pub-sub. There's like ZeroMQ and RabbitMQ and Redis. What were the problems with the ways that people were implementing the pub-sub pattern?

[0:04:38.0] SB: Good question. In terms of the problems, the implementation they used is fine. Their approach was having servers communicate with each other usually to do message brokering. You mention RabbitMQ. You've got a tone of other ones, like JMS, and you've got brokerless design patterns such as ZeroMQ. Those are server to server communication.

When you want to extend that beyond your data center and that phone that's in your pocket, the design pattern needs to change. It needs to be a little bit different. Those connections are not guaranteed like they mostly are inside of a data center. You're going to have different IP addresses throughout the day. You're traveling. You get the train IP. You go on 4G. You switch to Wi-Fi. You need a system that's able to sort of store and forward and follow that device as it changes its location in order to allow it to receive messages reliably.

[0:05:31.9] JM: Could you maybe like contrast and implementation of pub — I think this is back in 2010, although to some degree, it's probably still true today. If you wanted to use Kafka, or back then if you wanted to use ZeroMQ, these lower-level pub-sub abstractions, how would that differ from using what you've built?

[0:05:53.5] SB: Those systems are potential components within an orchestratable engineering system that we'd built with PubNub. Kind difficult to describe, essentially if you were to use a Kafka or RabbitMQ in your backend to facilitate at that store and forward mechanisms for devices outside your firewall, say, you wanted to create a publish-subscribe for mobile phone apps, you would start with, like, say, a Socket.IO or some sort of interface allows you to gain access to that message broker.

The challenges once you start getting a lot of subscribers and how do you federate or distribute the workload across many systems. We found the best way to do that was to write our own software, and we did this, we wrote it, and all of PubNub is completely written in C. It allows us to control the memory, the connection patterns, the retry, the store and forward or caching mechanism, the GC. It's a system that in order to scale to hundreds and millions of connected devices and have as many topics as you want, we had to build ourselves.

You look at Kafka, for example, fantastic system. Everyone loves talking about it. You have to tune that system to specific use cases. How big is the message payload size is going to be? How many topics are you going to have? How long does the data supposed to live on disk? If you have a specific use case, you can use Kafka and build your own infrastructure. However, once that use case starts changing, you're going to need to spin up a separate cluster specifically for that. That's why we built our own, because we wanted to sell it to anyone who wanted to use it with a simple publish-subscribe API call. That means it's a multitenant environment and everyone has different use cases. You mentioned earlier on, "Hey, it's not just chat as IoT devices." It's hailing taxicabs. It's making phones ring. It's ton of different use cases and they all have different load profiles on the network.

[0:07:44.9] JM: Do you think there's an apt comparison to something like Heroku, like a product like Heroku where they took what AWS was doing, and I think they did it around the same time that you got started. But it was kind of this model where, "Okay. We've got —" There are companies that are starting to offer pretty good infrastructure solutions to developers, but there is this other tier of developers that doesn't want to use the low level difficult to use solutions. Like what Heroku did, they took EC2 instances and made them easier to use. Basically they built APIs on them. They were simpler to use and they took care of a lot of the failure cases and they just made it easier to run your infrastructure. I think it kind of seems similar where you looked at this and you said, "Okay. Pub-sub is a very common use case and it's actually pretty hard to implement even if you have these abstractions," like the MQs, or the Redis or the Kafka. You really want kind of a full-fledged series of abstractions on top of that the just abstracts away everything that you don't want to think about.

[0:08:49.4] SB: Yeah, you nailed it. That's the most important part is the simplicity of implementation. You don't want to focus on building infrastructure as you're focusing on your

specific app. For example, if your Lyft and you just want to connect drivers and passengers to each other. You really don't want to have to orchestrate an entire federated Kafka cluster, that you'd much rather just build the application, make it better, make faster. You're competing with Uber. You don't have time to focus on building this messaging infrastructure. Especially, nothing like it exists as open source. So you're going to want to choose an existing vendor that can just make it really simple for you.

[0:09:30.7] JM: Yeah. You started this around 2010. What were the kinds of applications that people were building on it in the early days and how have those changed over time?

[0:09:39.9] SB: Oh! Games, gaming. That's even what got me into programming in the first place. It's like, "Look at these amazing things you can do with the computer. You can make the pixels pretty, shining, whizzing like particle effects," and just the essence of games and how they teach you things or they're social interactions.

Our first customers were multiplayer games on phones and like early phones, because back then it was just iPhone one, iPhone two type day. Android is barely getting started. A lot of web games. So that's when we got started. We started making a lot of money back then on games. We see chat starting to evolve. In fact, about 60% of our current revenue is coming from chat and a lot of the existing customer infrastructure that you've used that's probably on your phone right now is being powered by PubNub and you didn't even know it.

[0:10:32.5] JM: Fascinating. Yeah, regarding the games use case, we had the show recently about scalable multiplayer games and the challenges of building multiplayer games that work internationally, and especially on mobile devices where you get network partitions all the time and your Wi-Fi chops out, can be really difficult to figure out all the retry cases, especially because it's like it is a real-time experience, and like it's too bad when a Skype connection drops out, because Skype is kind of a real-time thing as well. But in a game, you might just lose. It chops out and there's no recovering from the network partition there. I can imagine people wanting to have some easier times building games assisted by some infrastructure.

[0:11:25.3] SB: Yeah, you nailed it. That's such an important part of that user experience you want, in addition to implementing that multiplayer stack into your app, you want your end-users

to have a fantastic experience. That's really the end of the day. I mean, for us, our customers are our developers and our customer's customers are those gamers. They're playing that game, and if they drop out and lose, they're going to have that bad feeling. The important part is not what do people remember. They don't really remember what you tell them. They remember how you make them feel.

So if you lose a game due to network connectivity, I mean, of course you're probably used to it on a mobile phone, but if you're just sitting there in your home Wi-Fi and it goes out and you know the Internet is working, you're going to blame the game and you're just going to have a bad feeling about that. We got to make sure that the experience is happy and joyful especially at the beginning.

[0:12:17.4] JM: You mentioned that the early product was built in C, and I'd love to know the rest of the early product infrastructure and then we can get into how it's evolved over time. So you basically built the pub-sub engine in C. Just give some more details on what that looked like, how it was deployed, how long it took you to build that initial version.

[0:12:42.2] SB: It's still C today. Our infrastructure is different than you would hear from pretty much any other infrastructure. We're globally distributed, which means that we've had to a creative a full mesh of C processes that are living in all the data centers that we deploy them to and they all connect to each other in some way or another.

Essentially, when you send a message through PubNub using the publish API, that message is replicated three times per data center and we copy the data. Every single data center gets a copy, and that allows us to have end-users connecting across nations, across countries to the closest data center. Then using interconnects between data centers, we're able to transmit those messages quickly much faster than you'd get if it was a peer to peer situation and much more reliably as well. That's what makes PubNub different. We're kind of like a CDN, content delivery network. You always connect to that closest data center, and that was our vision from the beginning. We always wanted to have that. In fact, we had around that specific model for delivery of service, and it's fantastic.

In addition, we have a specific engineering philosophy. This one is around not like a backup system. We don't want to have cold systems and standby that are ready to just pop up when other systems go down, because, hey, you know what? Systems fails all the time. Hardware fails. Connection fails. What we want to do is have a system that's always running and every server is always servicing traffic and we expects those nodes to fail.

We've built a system that allows that failure to occur and messages will get automatically routed around the failure. We make phones ring, and that's really important for customers, our customers, because every second that that phone doesn't ring, our customers lose revenue. We just need to make sure it always works and that's what we're providing.

[0:14:36.1] JM: That model of replicating each message three times on three different data centers, why do you have to do that? Why can't you just have the message replicated just on single time on each of those data centers?

[0:14:50.8] SB: It's a consumer-producer ratio and our infrastructure and our use case, we see that we have a 1 to 100 ratio of consumer-producer. There're a lot of consumers reading data off that same topic or need that message. So what we've decided to do is proactively replicate that. It's cheaper and more economical for us to replicate that message proactively even if someone else isn't going to be reading it in the other data center. So our costs are lower and our reliability is higher. It's kind of like a win-win, and you would think that, right? Why do you optimize it so that we don't send the message that no one is going to read? It turns out it's just not how people use a publish-subscribe systems.

[SPONSOR MESSAGE]

[0:15:39.5] JM: You're a successful developer and you couldn't have gotten to where you are without help in your education and career. Maybe you're thinking about ways to give back in the community where you live. The TEALS Program is looking for engineers from across the country to volunteer to teach computer science in high schools. Work with a computer science teacher in the classroom to bring development concepts to life through teamwork and determination. Pay your success forward by volunteering with high school students in your area by encouraging them on the computer science path. You can make a difference.

If you'd like to learn more about the Microsoft TEALS program or submit your volunteer application, go to tealsk12.org/sedaily. That's tealsk12.org/sedaily.

Thank you to the TEALS program for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:16:44.7] JM: I guess I'm still a little bit confused, because I can understand why you would want it on each of the data centers, but I'm just wondering why you need it three times on each data center. Because you can imagine, okay, user A is going to send a message to channel B, or channel X, and they send, they send the message to channel X and the hits a server, one of the servers, one of the three servers that holds all the messages for channel X, and then the message gets replicated three times on that server and then it gets sent to two other servers where it also gets replicated three times.

I guess I'm just confused why you wouldn't just keep the message one time on each of those data centers, and then if you lose the instance that's holding that message you'd just get it from the other data center.

[0:17:27.9] SB: Yes. It is. It's replicated three times per data center and, of course, to every single data center after that. We do that specifically for reliability. Now, if you look at the company TIBCO and may be other messaging bus companies, they offer what is called guaranteed messaging, a guaranteed delivery. They are able to achieve that at least in their claims and on paper by simply replicating a message twice. They replicate it twice, and that's what they call guaranteed message delivery.

We don't store those messages on disk. We store them only in memory, and this is a big reason why we want to do it at least one more time.

[0:18:11.9] JM: Because it's more volatile.

[0:18:12.6] SB: Exactly. So you get enough failures in a single data center to justify having those extra replicas of each message.

[0:18:20.3] SB: That's especially true, and there's one extra bonus. We have large events that come on to our network where they have high demand on reads or subscribes for that same topic and we need to replicate the data in each TCP packets to every single IP address requesting it. We need to be able to distribute that workload across our systems, and at times it can get spiky. Having that data proactively copied allows us to deliver on our SLA's of a quarter of a second.

[0:18:49.9] JM: Because you want to replicate that data so that you have more availability on the read side.

[0:18:54.6] SB: You got it.

[0:18:56.6] JM: Interesting. So maybe you can just describe what happens when somebody subscribes, or when somebody creates a channel, when they start publishing messages to it and how the consumption of a message happens, like, walk us through just to what happens on the infrastructural layer when that initial onboarding event happens.

[0:19:20.0] SB: All right. So let's paint a picture here. You've got a phone and you want to install a PubNub SDK on to the developer, we'll issue an API call within the SDK called a subscribe. They're going to subscribe to channel A. What that does is send a signal to our network. Before it does that, it does one important thing. Tt says which data center should I hit?

We use a DNS provider called Dyn. Maybe you're familiar with them. They had a DDay, October 20, 21st, a year or two ago. Most of the internet went down.

[0:19:52.9] JM: Yes. The Mirai.

[0:19:54.3] SB: Yes, that's it. You got it. That is a powerful — Actually, Cisco acquired them and believe they power Twitter, amazon.com. The power a ton of different companies, Cloudflare, Fastly, CDN companies and ours as well. What we do is that device will do a port 53 datagram

asking for the nearest data center. It will return IP addresses for the closest data center using Dyn's IP. It will establish a socket to our network that lives forever, essentially.

Our TCP policy is unlimited. So you have an always on long live connection to our network. You paste up the channel data and the last message that you want to receive, if you want all the data or just let me know if there's new stuff coming forward. That'll jump through our edge. It jump through some routing tables and then it lets you hit a specific node that will hold a file descriptor open and says, "Okay. This is file descriptor will get any information on this channel."

Next, we have another phone that shows up and it's going to be sending data. It uses the same mechanism of connectivity and then it will publish using the PubNub publishe API. That message hits our network, automatically routes the replication layer, hits all of the subscribe nodes internally and then that message comes back propagating down the original subscribe call.

[0:21:18.9] JM: Got it. By the way, your mention of Dyn, may be curious. What happened to your infrastructure when that Mirai DDoS — By the way, for those who don't know, this was — Basically, there were all these IoT devices throughout the world that had the same username and password default credentials and there was a botnet that logged in to all of those IoT devices that had those default credentials and started launching a DDoS attack against, Dyn, which his core internet infrastructure, and so the downstream effects was knocked down Netflix, knockdown Twitter, other victims. Did you have problems in that event?

[0:21:58.7] SB: Yeah. I was in contact with Cloudflare and Fastly and everyone else who is running on it. They were saying all we can do is watch. Watch the world burn. I was taking out my phone and I was Snapchatting all the graphs as they were shooting straight downward in this precipitous drop slope. I'm like, "There's nothing we can do." What we did, we tried switching providers and by the time it came back around, it started coming back. So it takes a while to figure out best. But what were you about to ask?

[0:22:30.5] JM: I was going to say, so at least we know Shapchat is not on Dyn.

[0:22:32.9] SB: Right, exactly. Yeah, it's Google infrastructure. They pay something of a $200 million or so annually to Google for that.

[0:22:42.5] JM: Wow! Were you on AWS initially? Because I know this was like two years after AWS got started. I guess back then it was a pretty real decision, do you go on the cloud or not?

[0:22:56.2] SB: For me, it was clear to go on the cloud immediately. I was just obvious after having experience myself in data centers and having to go visit that colo and I'll reboot the system or whatnot. Something that you don't want to worry about when you're starting a business, especially in tech today. It's just, "Oh! Yeah, press the button on Amazon. They'll take care of it for you." Even though they fail, they do, they fail, but they take care of the failures for you, which is nice.

[0:23:23.1] JM: Indeed. So it was not much of a decision, but did you have any scalability issues in the early days when you were onboarding all the customers early on and you just had your kind of your first version of your product?

[0:23:36.1] SB: Scalability, yes. Amazon wasn't able to handle our connectivity requirements. We had millions of devices connecting throughout their network, which had hammering on their firewalls and their network layer. We had to spin up additional data center capacity and what was called Softlayer. IBM had acquired them, other data center providers as well; DigitalOcean. We were able to continue to scale in Amazon as they improved infrastructure.

[0:24:03.8] JM: You had to have these other providers as overflow capacity, because why couldn't you just overflow on to additional AWS instances?

[0:24:12.7] SB: It was a core networking layer issue that they just couldn't handle with us. We'd see these massive one second global dropouts for every connected device. It's just that the way they had architected it was not set up for our type of connectivity. It should have mainly been Rest API calls is what they were good for and not long live connections.

[0:24:36.6] JM: So that's said, why didn't you — Did you consider like going more heavily on to DigitalOcean or on to Softlayer more aggressively early on? What was it about AWS that made

you continue to just, overtime, shift you're more and more infrastructure and less and less reliance on the other overflow providers?

[0:24:55.1] SB: Purely developer operations consideration. We have all been pieces we need, our orchestration was spun around AWS APIs. It made it easier for us to launch new data center co-locations with them. It was just such an easy — When Terraform, HashiCorp technology came out, it was easy for us hit not just Amazon, but any other — The providers that offered APIs to scale with them. So that allowed us to diversify. Yeah, which is great, and Amazon is still our number one data center provider of choice, because our customers are there too. We want to be right next to servers, which allows milliseconds of latency. If you're doing server-side publishing or if you're subscribing on the data stream, you should be able to receive that data locally.

[0:25:46.2] JM: You talked about most of the core application infrastructure of PubNub being written in C. It sounds like you are very comfortable building what you need to build. But was there stuff that you took off the shelf? Did you use some databases to the store the messages or do you have your own in-memory thing that you built to store these messages?

[0:26:07.3] SB: Databases, yes. We have — Our in-memory database is proprietary, which is based on our replication and our patents. We do have a disk-backed database that we offer through our storage and playback service called Cassandra. Perfect for time series publish-subscribe events. Boy! Did we have a ton of issues with that.

[0:26:28.7] JM: Well, you're not alone. I've heard that from other people. Cassandra can be difficult.

[0:26:32.7] SB: Oh, yeah. I was our fault. It was just a user — It's just like that Kafka thing that we're talking about earlier. You have to tune it specifically for the use case, and our customers are diverse. We tuned it for general use case purposes. However, I just didn't match. It didn't meet the requirements and we mixed their tables around and ultimately had a lot of downtime because of our misuse of the system. Now we're good at it. We've got professionals on site and we know exactly how to use it and how to scale it.

[0:27:02.6] JM: So you said you basically wrote your own databases though. You wrote an in-memory database.

[0:27:08.5] SB: Yes. It allowed us to have zero lock time on memory allocation and de-allocation. We didn't have any garbage collection pauses or anything like that with our system. It allowed us to be very nimble. Most of the messages that we send and receive are under 1 kilobyte, and trying to tune memory profiles for that became challenging, especially when diversity of a larger message payloads and the frequency changed and over larger message channels, a lot of channels. So many data channels. My goodness! We really got a great performance I'd ever seeing ability to sort of have a wheel or a clock tick, sort of tick away at the message memory, which allowed us to maintain our memory profile without having OOMs.

We used to have OOMs a lot of, a lot of "out of the memories". The kernel would just say, "Hey, this process is using too much memory." Boom! Dead. It just kills it outright, and that database would just be empty. So it just took us a few years to make that work without crashing. We've succeeded at that and it's been running fine for years now.

[0:28:15.5] JM: Did you write that in-memory database? Was that the first version of the product or did you use something else and build your own thing eventually?

[0:28:24.2] SB: We prototyped the infrastructure on, of course, Node.JS. Back in the day, Node.JS was just getting started. So it would crash regularly and it would have all sorts of fun challenges. As soon as we started hitting, scale and customers were knocking on our door. It's like, "Okay. We just can't keep sitting on these servers and terminal windows and keeping things up and running. We need to have some serious infrastructure." Then we just buckled down, wrote C using our globally distributed model, and that's the same code base. Of course, new iteration since then.

[0:28:57.6] JM: So as time has gone on, you've gotten bigger and bigger customers. Are there instances where you have a particular customer who pushes the limits of the infrastructure and then you have to re-factor the infrastructure because of that heavy customer?

[0:29:12.5] SB: Good question. It turns out it's a producer versus consumer problem. So if you've got a device or a server or something that's just massively publishing a lot of data per second down a data channel, that's fine. We like that. We're good at that. The challenge is the consumer on the mobile device or at a laptop isn't able to keep up. It'll start falling behind. It just can't keep up.

So what we've done is we've built in an automatic pressure system, which will begin intentionally dropping data. It's just, "Hey, you're pushing way too much data to this channel," and we're able to tune that on a per customer basis. It's automatic today and if a he customer has high-volume needs, we either help them or architect it properly or we tune a little configuration flag which propagates throughout the network within about a second.

[0:30:09.3] JM: When you have a new customer and they really do pusher your system, push the limits of it is, are there any adverse effects. Like does it cause some downtime or does it automatically rebalance necessarily?

[0:30:24.5] SB: Good question. It's automatically. The last thing that you said, that's the right part.

[0:30:29.4] JM: Okay.

[0:30:29.7] SB: Yeah. It auto-rebalances. Everything is in a streamline smooth. There is a perception though, and this is an important one, that not all the data will get to the entire device, that's because you're hitting our throttling limits. If you're attempting to either load past or just push the limits on that, you're going to run into our intentionally coded software limitations. We just like — We stop you at a certain point. We're like, "Okay. There's no way that other device is going to be able to receive that much data." So we're going to begin dropping those messages.

[0:31:01.8] JM: Oh, okay. Do you drop them completely or do you do hold them in some public like queue or something?

[0:31:08.0] SB: On the live connection, they're dropped. If you have storage and playback enabled, they're saved forever.

[0:31:14.5] JM: Got it. Do you have any other interesting edge cases that you've encountered, like difficult problems that you've had to solve as you've been building out the infrastructure over the last seven years?

[0:31:24.3] SB: We wanted to make it cheaper for us to operate, and we were talking about this multithreaded specialized memory system that would give us 60% cost savings on paper. We started working on it. It sounds promising. We got some initial good test results. Few months insulators, still same thing. Six months. All right, this has taken a long time. What's going on here? A year later, "All right. Let's see if we can get this thing into production," constantly crashes. It's just so many problems in a highly concurrent system like PubNub where we're currently doing 24 — I'm looking at the dashboard. 24 million transactions per minute, that building a threaded system is too difficult architecturally.

We have really smart people here, like low-level network. They know what they're doing. Just the challenge comes in with such a high volume of data and moving it around, having multiple threads all accessing same memory addresses just creates these uncertainties. It just didn't work. So we had to choose to throw away that years' worth of investment. Yeah, it was — And the promise, it feels like it's possible. It's still there. It's just too much code, too much complexity for us to be operating.

[0:32:49.0] JM: It sounds like when Amazon built the Fire Phone and they just had to throw it out eventually. But I think they got some value out of it. I think there were some learnings from that, for example, the like maybe carried over into the Amazon echo. Were there learnings that you took away from that experience that made it worth it or are you just like, "That was really not worth it and a terrible waste of time."

[0:33:13.0] SB: You always learn something when you fail. If you don't, then what's going on here? You know how I mentioned that we had a policy. It was like sort of no backup systems. Every system should always be hot and running to service the customer and expect [inaudible 0:33:24.4]. Yeah, we have another one. The other one is no threats, and it's not that we knew that it was going to be a failure, because it seems feasible. It's just the level of complexity that's introduced in such a highly concurrent system and it just reminded us to keep true to our ways

and just keep things simple. In engineering, you always want to reduce code. Code reduction is like bliss for development and it's like, "We don't need to maintain or own on that piece of code anymore," and from an engineering perspective, it's just happiness and comfort. So you should always go towards co-reduction and stray away from bloating your code base by trying to reduce costs.

[SPONSOR MESSAGE]

[0:34:19.1] JM: Users have come to expect real-time. They crave alerts that their payment is received. They crave little cars zooming around on the map. They crave locking their doors at home when they're not at home. There's no need to reinvent the wheel when it comes to making your app real-time. PubNub makes it simple, enabling you to build immersive and interactive experiences on the web, on mobile phones, embedded in the hardware and any other device connected to the internet.

With powerful APIs and a robust global infrastructure, you can stream geo-location data, you can send chat messages, you can turn on sprinklers, or you can rock your baby's crib when they start crying. PubNnub literally powers IoT cribs.

70 SDKs for web, mobile, IoT, and more means that you can start streaming data in real-time without a ton of compatibility headaches, and no need to build your own SDKs from scratch. Lastly, PubNub includes a ton of other real-time features beyond real-time messaging, like presence for online or offline detection, and Access manager to thwart trolls and hackers.

Go to pubnub.com/sedaily to get started. They offer a generous sandbox tier that's free forever until your app takes off, that is. Pubnub.com/sedaily. That's pubnub.com/sedaily. Thank you, PubNub for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:36:02.2] JM: What are the kinds of — Because I think that's actually a pretty good lesson and I think there's actually — I don't know. What are the cases where if you're somebody out there running a software company where you should pause and re-factor your infrastructure to

make it cheaper to run. Because it seems like in general, software is a high-margin business and you generally don't have to do that so much. You generally don't have to re-factor for cost. For example, like I heard a story about WP Engine recently. WP Engine basically runs WordPress. Just runs WordPress instances for people and they moved to Kubernetes and gave them some giant cost reductions.

Actually, even then I think they saved like 50% of their costs by moving to Kubernetes, which is pretty amazing. But in I think that the main reason they moved to Kubernetes was actually for operational reasons and the costs were just additional benefits. I don't know. Do you have any principles for when people should migrate infrastructure because of cost management?

[0:37:05.2] SB: The modern way to move forward with hosting and deploying in operations is these containerized management systems such as Kubernetes. We're doing this too. We're migrating all of our orchestration software, which we originally wrote in Python scripts and in Pearl scripts to manage how each of the nodes are interconnecting with each other, and Kubernetes just takes care of all that for you. It makes it so much easier. Your cost on manpower goes down. We've noticed that we don't have to deal with certain things anymore. We have half of our infrastructure on it now, especially when you have many different computer patterns, they're not homogeneous, right? They're essentially this — You've got one kind of process that takes up a lot of RAM. This other one is very CPU hungry. This one uses a lot of network. What you get to do with Kubernetes is sort of shove all those different compute patterns into a single box. You can leverage what you're paying for. You're our paying for this EC2 box that has all these capabilities, 10 gig network, a bunch of gigs of RAM, little bit dik, some CPU. Our current workload profile for some of our systems only uses one of those things, and then bottleneck is hit.

You don't want to have to code to like your application to use all those things if it can't, because just how it works. If you could instead just throw other workloads at it, be a Kubernetes, it knows how to use distribute that and give you the best bang for your buck. We've found it gave us the same thing you're describing, fantastic cost savings.

[0:38:44.1] JM: You've been managing this company now for about eight years. How is the management role evolved over time as the company has changed?

[0:38:52.7] SB: There's this Andreessen Horowitz a16z that always talk about the different phases of a startup's lifecycle. The way intercommunication happens, the way agile works, you need to get more streamlined. It's just as we grow and teams expand and need to have their own initiatives, communication is just totally different now. Every company's solution is a meeting, "Oh! Let's just have a meeting for that." After a while, you just try to keep solving with meeting after meeting and then you have that, "Let's have a meeting about meetings." It just becomes this horrible experience we have, especially in the engineering team. You want them to say, "We need more meetings." You don't want to be in the place we're just sitting in meetings all the time, we're not getting anything done."

Of course, we still talk about it. We still say, "Hey, we're not optimal on our time when we're meeting the other." A lot of people are just on their laptops, like not paying attention. So how do you solve that? That's a challenge. What have you heard from other business? How do they solve that meeting challenge?

[0:39:59.9] JM: The first thing that comes to mind is I had a conversation with someone from Box recently and he was talking about something similar to this, but it sounded like the way to solve communication from his point of view was, first of all, you do unfortunately have to build layers of management hierarchy. It's just a necessity. We could talk all about holacracy or like totally flat organizations, but in general you're going to solve this with hierarchy.

But then you push decisions to the bottom of the hierarchy as aggressively as you can. You let the people who are actually implementing things make the decisions as much as possible and you just set a strategy. The people who are anywhere above the bottom layer are just sort of setting strategy and letting people operate autonomously as much as possible. I'm sure that on certain circumstances you have to be a little more micro-managerial, but I feel like that's been a theme that I've heard, this like it's sort of decentralized, pushing decisions out to the edges of the organization.

[0:41:03.0] SB: Absolutely. We subscribed to that as well. That's critically important. You don't want to be — That's kind of hard though. Sometimes you want to be there because you think know what's right and what makes sense. However, that's not where the value is. The value is in

your team and the company itself. That team needs to be self-sufficient and move forward, and big important topic is ownership. They can't own something that they haven't decided on. Its motivational. If they're able to say, “Hey, I bet my job that this is the best way that we can go.” They’re going to ensure that that is carried forward, and that sort of motivation is hard to get if you're barking orders, right?

[0:41:43.3] JM: Have you had to do anything interesting to keep those communications running smoothly as that headcount has increased? Have you developed additional management layers or any kind of particular practices that you've codified?

[0:41:58.0] SB: We tried to be inventive and do things our way, and we started to notice just this is a solved problem. If you look at larger organizations and their structures, it sounds bureaucratic and boring. It turns out it's that way, because it kind of works. You need to have the team that you're describing. You need to have the upper layer of management that's able to enable their teams and their department to be successful and push them forward and being motivated. It's really that’s simple. You can't be too inventive with this. You might as well just use what works.

[0:42:32.6] JM: Sorry. You were referring also to things like key performance indicators, and I guess SLAs and any other acronyms that you put in place or — Are there books that you've read that outlined particular management philosophies that have been useful at PubNub?

[0:42:51.1] SB: So we’ve tried doing some of those things. We've tried doing OKRs, which we currently are. We’re doing the Google style OKR. It's been rough — Most team members say, “Hey, if we’re listing out all of these objectives over the quarter and the goal is to only get 70%, why is it only 70%? Why can’t it just be 100%?” It didn’t make any sense. Of course, we’re still struggling with some of that. Now we drive all of your teams and all of our business on something that’s measurable. It has to be measurable in ways like, “Okay. Our uptime needs to be five nines.” If we dropped, then we know that we didn't meet our objective and we need to give credits to our customers.

Our inbound leads need to be X-number of sign-ups per month. We need to keep pushing and growing that. That number needs to keep going up. Being a marketing or a metrics-driven

organization helps a lot from a managerial perspective. From a team perspective, how do you know when you're being successful? How do you know in this day and age where numbers are really big and unimaginable, even the number 1 million? It's just like one or two off from that is non — It's like just not meaningful.

So if you're able see larger trends over longer periods of time and measure that, you can see trends. It's like stocks, right? Easy numbers. You see when the Dow Jones goes up or Dow Jones goes down. We need the same thing for running the business, and it just makes it easy for us to celebrate, and then when we lose, we just figure out how to do it better.

[0:44:25.7] JM: Do you have an SRE facet to your organization or some other kind of — Basically, how do you do incident response and on-call and monitoring and logging. Those philosophies that generally get bucket under the site reliability engineering role?

[0:44:42.6] SB: PagerDuty.

[0:44:45.4] JM: Yes.

[0:44:45.9] SB: Easy one answer. Yeah, we have site reliability engineers in each team, multiple teams here at PubNub. The teams are responsible for their pieces of the infrastructure and they share a pager. Now, the cool part about most of our infrastructure is it's automated and will self-heal even if Amazon goes down. Our SDKs that are installed on mobile phones will connect to the next closest data center. For the most part, there's not a whole lot of operations and when stuff goes wrong, it's self-healing and customers don't even notice, which has been pretty fantastic for us.

[0:45:24.4] JM: So what about monitoring and logging?

[0:45:27.1] SB: Yes. So that's very expensive for us. It's about 3 petabytes of messages, JSON messages traverses our network every month. We used to save all that data. It became really expensive. Then we turned on this — We were saving into Amazon S3. Then we turn on this thing called Glacier, and it saved us a little bit of money. It just kept growing and growing. Now, it's just too expensive for us to save that. So we actually just dropped the data and it just goes

away, which makes me so sad. There's these like TensorFlow and artificial intelligent machine learning things. We've got data here that is extremely valuable we could drive information from some sort of meaning and we're throwing it away. Darn! We have to though. It's just too expensive to keep it all.

[0:46:13.5] JM: Wow! One managerial challenge that I thought might exist for this company when I just kind of looking at the nature of your company, was you've got a lot of APIs and you also deals with other API companies. You have these abstractions that you've built on companies like Cloud Neri and Clarify, and then you have to build a lot of SDKs. So like you have SDKs for web, then mobile phones, and IoT devices, and if I think about the matrix of all of the different customers, it's a lot of surface area of different ways that people can use the products together. I wonder, does the surface area of all those different use cases ever become hard to manage or do you have any systems that you have in place that make it easier to manage those?

[0:47:03.7] SB: Yes. The age-old platform support question, which platform to support and how expensive this is? Very expensive. For us to support these mainstream platforms, we have teams that have expertise in each of these. It's important for us to make our customer successful using techniques in each of the platforms that allow extreme liability. We have to build in the automatic data center, switch over into each of the SDKs. Each SDK needs to know what messages is received and which ones it needs to fetch next. Yes, you're right. That's just a massive footprint. However, how else would you do business if you need to make it easy for everyone? There's just no other way to do it.

[0:47:49.9] JM: Definitely. Do you have like testing mechanisms in place or does it just remain pretty consistent across the — When you have to build a new SDK for web and mobile and IoT, like you have a new — You build these new services, for example, the chat, the new chat engine service that you built. If you wanted to have an SDK for the web and the mobile and the IoT devices that all interfaced with this chat engine, would you try to write tests that work on all three of those SDKs or would you write specific tests for each of those SDKs? Like what are your principles around testing the different places where people can be using those APIs?

[0:48:31.8] SB: All right. A quick easy answer, we have a book or a Google Doc. The Google Doc says every SDK must hit these criteria and have these tests implemented. So it's really simple and we have to just follow that rule and make sure that it subscribes to channel A. It has this —

[0:48:50.6] JM: Acceptance tests.

[0:48:50.7] SB: Yeah, exactly, acceptance tests. They must be runnable continuously in a CICD framework. So we have Travis, for example. It continuously runs connectivity tests across all of our SDKs including long lived tests and short-lived quick tests to make sure that the data is what it needs to be. We have a subscribe API. However, it's really complicated based on all the different capabilities that an API can ask for an expression to filter out certain messages via our server. It can ask for a channel group subscription, which can receive a variable and changing amount of data based on what's in that channel group. There're a lot of tests and a lot of variances that — I believe there's around 200 and some tests that must be implemented for each SDK.

[0:49:43.2] JM: Fascinating. As we talked at the beginning, the core use case that you build everything else off of was this data stream network that allows people to have pub-sub channels. So the basic idea of pub-sub is so generally applicable that you can actually build a lot of other products on top of that. So how do you think about the product expansion? How have you thought about what products to build on top of that core pub-sub infrastructure?

[0:50:13.8] SB: It's customer-driven. So our customer asks us, "Okay. We've got this cool publish-subscribe API. The data gets to the mobile phone, and it works. We're happy with it." It would be great if we could tell, is that phone still online? Did it go off-line? How can we tell when this happens? So we built a TCP layer infrastructure that allowed us to detect the connectivity of that device best as we can, of course. We call that the PubNub Presence, and they asked us, "Wow! I'm sending all these messages to you and I don't have any way to retrieve them later. How can I do this?" We built storage and playback, which is that Cassandra thing I was telling about earlier.

Then our customers asks us about these weird crazy requests, “I need to be able to count this number and updated badge number here,” and blah-blah-blah-blah-blah. How do we actually turn that into a product? What the heck do we do? We couldn't consolidate all the customer requests into a single next phase product.

So you've heard this serverless race phrase, right? Serverless — It’s all these. We had to build a way for our customers small bits of logic, a couple lines of JavaScript, into our network to tune or follow their business rules on that data stream and that data feed either to augment the message, block the message because of contents, insert additional content using HTTP rest calls to throughout the services. PubNub functions is the product we built to support that, and those functions, when you deploy a line of JavaScript with like a single if statement that just says, “If there's the word Coca-Cola in this message, then block it.” That code is distributed to every single one of our data centers, and any time a message is published to one of those data centers, that code is run locally at that data center.

[0:52:04.3] JM: This whole serverless trend, there's a lot of different companies that are building this functionality into their products. I have talked to CloudFlare, of course, Amazon, and then Auth0. How are you behind this trend of people increasingly building in this serverless functionality that allows the customers to write code into the higher level APIs that you wouldn't have expected customers to want this kind of thing in the past.

[0:52:34.5] SB: Right. It's just one of the things that rolled over and the trend kept happening. If there wasn't a concept called serverless, we would still want to go down this path that we went down. Being able to give your customer the control of how the network is involved and shaped is very important, especially since you need somewhere to host your trusted code. You want that code to execute in a trusted environment, not on a client mobile device, which is uninterested code execution, because those things are hackable and crackable. You want it to store your business logic in a place that it just can't be tampered with.

Typically, our customers would spin up their own servers. They’d just be like, “Okay, let’s spin up a Ruby server or a Node.JS server,” and then put our logic there, and it would have to sort of do this ping-pong hop, which introduced latency and a single point of failure. So we decided, “Oh, we can't have our customers keep using this broken design pattern, especially since our

network is globally distributed and they're basing it off of this one process. If they have hosted in a data center, that could just fail and their entire product fails, and it looks like PubNub is broken at that point." We had to build a globally distributed system that automatically updates any of the JavaScript code that you want to push to us even if it's a small one line if statement or a full-blown node app.

[0:54:00.0] JM: Fascinating. So what are some of the other APIs that you've built on top of this core networking stack? Because I know you've got that chat engine thing that lets people build chat systems more easily. It's just like a host of other higher level products that you've built. So what are some of the other ones that you've built on top of this?

[0:54:19.4] SB: We have a visualization framework called Eon, and it gives you charts and graphs and gauges that move in real-time that uses D3 to animate on your web browser. It's great for creating dashboards or visualizing the data stream on PubNub. We have a web RTC SDK that allows you to connect between two [inaudible 0:54:39.1] addresses to establish a peer to peer connection. Those types of products that we built are sort of like opinionated frameworks, because with PubNub, you can use any sort of channel topology that you want. You can create channel A, B, C, D, E, F, G and you can also do A.*and you can have channel groups to have a list of channels that are being subscribed by the device. It's really powerful all the capabilities and features you have, however, it's easy to make a mistake in your channel topology decision. So what we did was we saw our customers wanted to implement chat. They wanted implement a dashboard.

What we did was create an opinionated framework that follows pretty much all of the designs that are required for those systems and implement it in a way that a customer would be able to adopt off the shelf really quick.

[0:55:27.0] JM: Steve, it's been great talking to you. I want to wrap up by just talking a little bit about the future. So what are the future products that you're thinking about and the improvements to the core infrastructure that you're focused on?

[0:55:37.5] SB: So we talk about real-time, and that, of course, real-time would be like instantaneous, like perfect, like boom! It happened. However, our universe is currently held back

by the laws of physics and the speed of light is the fastest that you can go. I would like — I've done research on how to do this for quantum entanglement where it's like spooky action at a distance quoted from Albert Einstein. Is there a way that we can have true real-time communication? Of course, I'm talking a little bit of sci-fi here, but that would be pretty darn silly for us to help come up with a solution.

However, after doing some research, it turns out the current quantum entanglement principles are not used for communication. They're used for cryptographic capabilities. Essentially, you can entangle two atoms, two particles, they're essentially simply spinning at the same like axis now. You can then encapsulate those in like boxes and then move those boxes far apart. Then you can ask or you could probe that particle with a question; what's your current integer value? Probing one of them and then other one at the distance will give you the same identical integer value, which is great for shared secrets, then every iteration after that. Of course, it decays after a while. However, that's what quantum entanglement currently is. I wanted to be the real-time communication though. It's not that.

So if we're going to talk about some real-world stuff here, you know how we built —

[0:57:08.6] JM: No. Let's talk more about fancicle real-time communication problems and solutions.

[0:57:13.6] SB: That's really where the cool stuff is. We are building like chat engine, whereas opinionated framework for chat to make that really quick and easy, which panned out and made it really easy for customers to get live quicker, which brings us revenue faster and brings our customers to market more quickly. We're going to do that with IoT as well. We have customers like Logitech and Samsung, and Wink and Vivint. They have IoT devices such as a smart fridge and a home automation kit where you can turn on and off your light bulb from your office. So it's really easy for you to do those things with PubNub.

We're going to create an IoT framework. I don't know what it's going to be called yet, IoT engine, who knows? I'm going to do the same thing with videogames. Gaming is even more difficult, because you have latency mitigation strategies that you need to implement to give the

player a good experience. We have to implement and build those, specifically for unity is our target.

[0:58:09.2] JM: It's funny, because IoT is one of these things where it's just like every year we must be getting closer to that bright IoT future. But you could look 10 years ago and people were building IoT devices and IoT frameworks and stuff, unfortunately we don't have widespread adoption of internet connected fridges and internet connected light bulbs and so on. But at the same time we know that it's coming. That kind of stuff is going to come eventually. It's just one of those things where you have no idea when it's actually going to come.

[0:58:40.2] SB: Exactly. Yeah. We're living the world now. It's happening.

[0:58:43.7] JM: Yeah. Anyway, Stephen, thanks for coming on the shows. It's been great talking to you.

[0:58:47.2] SB: Yeah, absolutely. It's great to be here today. Thank you, Jeff.

[END OF INTERVIEW]

[0:58:53.3] JM: We are running an experiment to find out if software engineering daily listeners are above average engineers. At triplebyte.com/sedaily, you can take a quiz to help us gather data. I took the quiz and it covered a wide range of topics; general programming ability, a little security, a little system design. It was a nice short test to measure how my practical engineering skills have changed since I started this podcast. I will admit, although I've gotten better at talking about software engineering, I have definitely gotten worse at actually writing code and doing software engineering myself.

But if you want to check out that quiz yourself and help us gather data, you can take that quiz at triplebyte.com/sedaily and in a few weeks we're going to take a look at the results and we're to find out if SE Daily listeners are above average. And if you're looking for a job, Triplebyte is a great place to start your search, fast tracking you at hundreds of top tech companies. Triplebyte takes engineers seriously and does not waste their time. I recommend checking it out at tripledbyte.com/sedaily. That's triplebyte.com/sedaily.

Thank you Triplebyte for being a sponsor.

[END]