EPISODE 1170

[INTRODUCTION]

[00:00:00] JM: Banking and money management are at the core of many modern applications. Payment operations teams work to enable the transfer of funds between different bank accounts and to track the movement of those funds effectively. Modern Treasury is a company that builds payment operations APIs. Sam Aarons works at Modern Treasury and joins the show to talk through the engineering at Modern Treasury.

If you want to support Software Engineering Daily in a greater capacity, go to softwaredaily.com and become a paid subscriber. You can get all of our episodes without advertisements.

[INTERVIEW]

[00:00:37] JM: Sam, welcome to the show.

[00:00:38] SA: Thanks for having me, Jeff.

[00:00:40] JM: When I make a payment online there are a number of different hoops that are jumped through to have that payment be processed. Could you give me an end-to-end description of what steps are taken when a payment flows through the online system?

[00:00:58] SA: Absolutely, I like to break down this step into really three distinct steps. So there's what I call payment initiation is the first step. So it's actually sending that payment instruction to the bank. So let's say, Jeff, let's say I'm sending you a dollar and I'm in business, I'm Modern Treasure, I'm paying you a dollar. Payment initiation is about sending that instruction to the bank to move that one dollar into your bank account. And so that's the first step of every payment system. A lot of us are familiar with going onto our bank portals, entering in an account and routing number and an amount and clicking send. That's how we

do things manually today. And then there's a second aspect of payments, which I would consider reconciliation.

So after I send you a dollar, Jeff, a dollar debit is going to hit my bank account. So I'm going to see a dollar debit in my bank statement and it's supposed to come with a description that's human readable, but I think you and I both know it's mostly bank readable. So I as a consumer and a business have to figure out what does this one dollar debit correspond to, and reconciliation is the process of matching that debit to the original payment instruction. And so that's what I think of as reconciliation. And then finally the third aspect of payments I think is accounting. So taking that one dollar payment and actually saying, “Okay, I paid Jeff because I owed him a dollar,” because I lost a bet, for example. So accounting is saying this dollar has meaning that this dollar has purpose.

And for a business, you put a general ledger code, for example, and you put that into your general ledger such as a QuickBooks or a NetSuite instance. And so I think this is really the flow of payments as it exists today and this is really an abstract view of it. But essentially any business paying another business or consumer has to go through these three main steps of payments.

[00:02:51] JM: Are there any of those steps that suggest that this payment system in its current form is antiquated or difficult to work with?

[00:03:00] SA: I would say all three aspects of the payment pipeline have antiquated aspects to them. The first being the payment initiation piece. What I described previously was essentially an accountant logging into the bank portal or a person logging into their bank portal and entering in information by hand. If you're a business doing multiple payments, let's say 500 payments or more per month, it becomes rather tedious to do that by hand. And what you really want to do is you want to automate those payment instructions. And right now, essentially, businesses have a few options. One, they could try building their own system in-house to automate those payments, or they can use something like modern treasury and just have that integrated from the get-go. And so what we do is we provide a very simple

system for automating those payments and making those payment instructions as well as doing the reconciliation and accounting for those payments.

[00:04:03] JM: Tell me a little bit more about what Modern Treasury does.

[00:04:06] SA: So Modern Treasury is a full stack payments platform and we handle the three main pillars of payments. The first being payment initiation. The second being reconciliation, and the third being automatic accounting. And so Modern Treasury takes all three of these steps and streamlines the entire process. The way that traditional systems work today is they treat these three individual steps as completely different systems. So a typical business might enter in payments by hand into their bank portal once a month. So an accountant might have a hundred or so payments that they have to make. They log into the bank portal. They initiate them by hand in the bank portal. And then at the end of the month they get a printout of their bank statement. So the individual credits and debits that were posted to the account and the accountant will go through each of them one by one and try to figure out which one matches to the payments they made throughout the month. That's what we call reconciliation. And the third part, the accounting aspect, is where they take all of those hundred payments and assign a specific code to each of them.

And so Modern Treasury doesn't treat these as three individual steps. We treat it together as one unified process. So with Modern Treasury, at the time you initiate a payment, you tell us how you want it to be booked into your accounting system. And then from there we handle the transmission of the payment to the bank. We handle the reconciliation of that payment and we take both of those things and we export it into your general ledger system with the correct code. So it's really streamlining the process of payments. For customers who use us, there really is no such thing as an end-of-month close or an end-of-quarter close. Everything is being handled automatically by Modern Treasury. There is no manual reconciliation and there is no manual end of month close.

[00:05:55] JM: So what kind of businesses is Modern Treasury for?

[00:06:01] SA: That's a great question. Really, I think the unifying factor amongst all of our customers is that they move a lot of money in and out of our bank accounts. And so when you look at it from that dimension, there's a wide variety of industries that modern treasury appeals to. So we do really well in insurance tech companies. We do really well in fintech companies. We do really well in healthcare companies. We do really well in neobanks, and we do really well in platforms and marketplaces. And like I said, the unifying factor there is they take money in, they perhaps take a cut or they do some processing on that payment and then they send it out again. So any customer with a platform or a product that moves money is a really appealing customer for Modern Treasury because they have this problem of initiating payments, reconciling them and accounting for them. And this also is independent of customer size. We have customers from the three-person startup all the way to the multi-billion dollar listed public companies. So really that unifying factor is they move a lot of money.

[00:07:07] JM: Tell me about the canonical technical problems of building Modern Treasury.

[00:07:12] SA: Yeah. I'd say there are two major technical problems that Modern Treasury fixes or takes on. The first being item potency. So in payments, you never want to send the same payment instruction twice. I'm sure the receiver would love to receive the same payment twice, but the business sending the payment probably likes that situation a little less so. So one of Modern Treasury's main technical challenges is making sure that we execute payment instructions at most once. So whenever we write a new payment handler or we're integrating with a new financial institution, we want to make sure that the payment instructions or the files or the API calls, whatever that interface is, we want to make sure they're happening exactly once. Never twice, because we don't want to double remit payments. So we've made a huge investment in making sure that a lot of our systems fail cleanly. And when they retry, they don't double retry the same payment over and over again. So that's one of the main technical challenges.

The other technical challenge is just the asynchronous nature of payments. I think a lot of tech and API companies are so – Everything is synchronous. You make an API call, you get back an immediate response. And that's really not how payments works. A lot of payment systems are

predicated on the idea that you send a payment instruction and then you never hear about it again. It's a little bit different than what most engineers expect. And so you have to build your backend architecture in a very asynchronous way.

So when you send a payment instruction, you're not going to hear about that payment instruction for maybe 5 minutes, 10 minutes, sometimes even 24 hours. So you have to build a system in such a way that can take in information at different times and also send that information at the correct times because of batch timings and cutoffs. There're a lot of problems there. And so you have to make sure that as things happen, you're reacting to it, because really there are no synchronous interfaces in banking.

[00:09:16] JM: How does that affect the programming language decisions that you make?

[00:09:21] SA: That's a great question. I think it more affects how we architect things on the backend. Just to give a little context about our technical stack, it's all Ruby on Rails. We're hosted in AWS and we have JavaScript and React on the frontend. It doesn't really affect our choice of programming language so much. But what it really does affect is how we architect the internal components of our system. So essentially what that means is we rely on a lot of background processing. And so for Ruby, I think some of the best background processing libraries available are – Well, one of them at least is Sidekick. And Sidekick is a wonderful background job processing system that uses Ruby as well as Redis. And so we architect a lot of our systems around Sidekick and making sure that we can just enqueue a job and have that run at some point. And because we're hosted in AWS, we use a liberal amount of SQS queues. So queuing up messages inside of AWS and dequeuing them for further processing later. I think we more model – We chose the programming language first and then we chose the architecture around it and we didn't let the architecture dictate our programming language.

[00:10:37] JM: Gotcha. So tell me more about an example of an asynchronous workflow that you've built.

[00:10:43] SA: Absolutely. So I think the idea of payment initiation is the Uber or the ur example of an asynchronous workflow. So I'll give you an example. At many of our financial institutions there are no APIs. We're implementing fixed width text files that we send over SFTP, and there's a variety of different file formats that that can go in. But an example is let's say that I wanted to pay you, Jeff, and I'm re using that same example. I want to pay you one dollar. And let's say that the bank told me that I have until 7PM today to send you that file. So what Modern Treasury does is it holds on to all the payment instructions it receives throughout the day. And then right before the cutoff time of 7PM, Modern Treasury will generate a fixed width text file and we'll send it over to the bank's SFTP system. And then we don't hear anything about that file.

So the way it works is after we send the file, we don't get a confirmation, we don't get an acknowledgement. What we do get the next day is a list of the credits and debits that were posted to that account overnight. So what we can do is we can scan those credits and debits that come in and see if our payment is in there. And that's how I know, or that's how Modern Treasury knows that that payment completed successfully. And so it's a very weird asynchronous workflow where you're sending a payment file, and in a completely different unrelated payment file you're getting an acknowledgement, but it's not acknowledging directly your payment file. It's acknowledging all credits and debits that were being posted and you have to figure out which one of those is yours. And that pattern happens again and again at different financial institutions with varying degrees of different behaviors and workarounds. And it's an extremely challenging technical problem. And really, you just have to keep working at each bank and understanding each bank systems to get it working 100%.

[00:12:51] JM: There's been a lot of other payments companies over the last five, ten years; Stripe, Plaid, uh many others. Why wasn't this problem of money management – Why wasn't it solved before Modern Treasury?

[00:13:09] SA: That's a really great question. I think one of the things that's been happening in the background of personal finance or fintech in general is the explosion of bank-to-bank payments. I think 10 years ago we were seeing a lot of investment being put towards making

sure that consumer payments were in a place that could be better improved by providers. So if I take a step back, if I look at the Stripes and the Braintrees, and the audience of the world, they've done an enormous investment into helping companies process credit card payments, and that space is about – I think consumer credit card payments annually I think amount to about five trillion dollars a year. But business payments, bank to bank payments are about 18 and a half trillion dollars a year. So it's about 4X bigger and it's greater than 75% of the market. And I think for a lot of companies who didn't engage directly into bank to bank payments, you just wouldn't know that this was a problem. And in the past 10 years I think we've seen a lot of more fintechs realize that beyond cards there's a whole other world with regards to payments. Payments is a very big space. And bank to bank payments in particular didn't receive that same level of investment in making the interfaces easier. And so Modern Treasury is coming in at a time when a lot of fintechs are looking to make the experience of paying someone or receiving a payment much, much simpler.

[00:14:47] JM: You mentioned the connection between Modern Treasury and the rise of these challenger banks. Can you tell me more about what changes challenger banks have brought to the banking industry?

[00:14:59] SA: Absolutely. I would say the biggest thing the challenger banks have brought to the fintech industry is the – And I don't know a great phrase for this other than the democratization of checking accounts. Really, these challenger banks are great at allowing individual consumers to open up checking accounts. They potentially can get debit cards linked to those same checking accounts and just offer consumers a low cost or even no cost way of getting access to the financial system.

So I've seen a lot of challenger banks issue ACH routing and account numbers to their customers as well as a debit card. So with those three things, a user of that challenger bank can go ahead and get their payroll directly deposited into their bank account. And then upon that happening, they can immediately use that issued debit card, go to the store and buy something. And so it's allowed a huge amount of previously underbanked or not banked individuals to have access to the financial system in a way that I just don't think was happening

before with the non-mobile native solutions. I think a lot of these challenger banks thrive on the idea that they exist purely as a mobile application in many cases.

[00:16:20] JM: How do you test this kind of high-volume payments infrastructure?

[00:16:25] SA: That's a great question. One of the main things that Modern Treasury does, at some of the financial institutions we integrate with, we actually open up our own bank accounts. So let's say I wanted to integrate a certain financial institution and that financial institution was large. If I wanted to make sure that our integration was rock solid, what Modern Treasury will do will be to actually go to that bank and open up a small corporate modern treasury account. And so we go through the same steps that our customers would normally go through to open up accounts at those same banks. And in doing so, we not only learn how those banks work, we also learn what questions our customers are going to get asked. We learn what paperwork is going to get sent their way. And then when it gets time to integrate with those banks on a technical level, we get sent all the documentation that we assume would go to our customers if they didn't have something like Modern Treasury in place.

And so we use our own money in some instances to test out and make sure that these bank integrations work. And that's been a phenomenal way of just making sure that we really have a production handle on how these financial institutions behave and it's allowed us to fix an enormous amount of bugs before production as well as to figure out what we call reconciliation to figure out how the credits and debits are actually posted to the bank account and how we can associate those back with the original payment instructions. And so if we have even just a few days to play and test around with that before letting our customers have access to the same systems, it really gives a ton of confidence into the tools that we're building. And, really, stability is key for something like Modern Treasury. We want to make sure that we have stable interfaces that work correctly and never change and opening up bank accounts on some of these banks is a great strategy for achieving that.

[00:18:21] JM: What's the difference between what startups want and what large enterprises want out of a payments platform like this?

[00:18:29] SA: That's a great question. It really boils down to what we see – It really boils down to controls and reporting. So the smaller three-person startup, there's only three people. One of them might be in charge of finance for the whole company. There're not a lot of approvals and controls at that level. So any payments that a person or an API key creates through Modern Treasury, those payments are likely to get automatically approved and sent off to the bank, because that's what the three-person startup wants. But we don't see that same behavior at the corporate level. At the corporate level, you have multiple approval levels and multiple different groups of people that can approve different payments depending on what the payment is. And so at the corporate level, we see customers creating their own approval rules and their own approval hierarchies and then figuring out internally which payments should be approved by which group. And so they care a lot more about making sure that everything is audited, that there are controls and approval rules in place. And Modern Treasury is flexible enough in that area to provide services to both the three-person startup and the corporate user.

And then the other aspect I mentioned was reporting. At the three-person startup, they may do reporting slightly haphazardly. They're growing fast. They're making a lot of payments. They'll check reporting when they need to or they'll check the reports if something is askew. But otherwise they don't keep too close an eye on the payments that are being initiated. Whereas in a corporate customer, reporting is front and center a problem that they want solved and something that they need tools for. So for our corporate customers, we have a lot of reporting infrastructure for allowing our customers to dive in and see individual payments, see different patterns, collect a lot of details and really download CSVs. Every customer loves CSVs. All the pages on Modern Treasury allow you to download a CSV of payment data. That's just a standard feature. And because we offer that, those same tools are also available to the three-person startup, but they care about reporting just a little bit less.

[00:20:42] JM: Tell me more about the APIs that you want to expose to users.

[00:20:47] SA: That's a great question. Can you clarify a little bit more about what you mean by that? We expose a lot of different APIs. I just want to make sure I answer the question correctly.

[00:20:57] JM: Yeah. Just like what the surface area of APIs is and how the average developer plugs into that.

[00:21:04] SA: Yeah, I can definitely speak to that. So I've been saying it again and again during this interview, but the first thing though, the first step in any payment operations workflow is the payment initiation piece. And so the primary API that a lot of our users have their first experience with is what we call the payment orders API. And that API is just for, like its name implies, creating payment instructions to send to the bank. So we aim to provide APIs that abstract away the underlying implementation details of the bank and the underlying complexity of how Modern Treasury communicates with that bank.

So I really go back to first principles. I think about it as myself as a developer. I think, “Okay, what are the APIs that I would like to use? Or what are the APIs that I would like to consume?” And in many products, and this is no different from the payment orders API, we think about these things from our own perspective and we build the interface that we want to see. We build the interface that we would like to use and we design the API around that and then we let Modern Treasury handle the nitty-gritty of actually talking to the bank, actually doing the reconciliation and actually doing the accounting.

And so in terms of surface area, we try not to expose a lot of different functionality and features through our API, but we expose the big things. We expose creating a new payment. We expose getting a list of the credits and debits that have already been reconciled to your payment orders, and we offer you the ability to see things such as counterparty information. So storing your users account and routing information inside of Modern Treasury.

I usually tell prospective customers that if the bank offers a system for something, we'll build an API on top of it. So Modern Treasury's API surface area really mimics the underlying

functionality of the banks that we integrate with. So we really try to make a system that applies to all the banks, really, in the world, but primarily in the United States. And then from that, we do the de-normalization of making sure that that API translates well into a form that the bank understands and the bank wants to receive.

[00:23:28] JM: How are your engineering teams arranged?

[00:23:31] SA: So right now we have a flat reporting structure. So all the engineers report into me. We have a pretty small engineering team right now. Right now we have seven engineers and myself, so eight engineers total. And right now we work in a very siloed fashion. So whenever a new product or a new feature comes through and we decide who gets to work on it, I often just assign it to any engineer who's available and who's free and then they work on that feature by themselves entirely. And if they need help, everyone can always give them help. But by and large, features and products are delivered by a single engineer. And because that's the case, all our engineers are full stack developers. So they can do frontends and they can do backend. Everyone has slightly different preferences and strengths. But really the engineering team we have today is as cross-functional. And so I trust any member of the team to take an entire product or an entire feature and deliver it end to end.

I think in the future as the team starts to scale up we'll start splitting the team into more specific areas. We've talked about this a little, but I think a potentially good split for the team in the future would be between what I call applications and infrastructure. The names are still being worked on a little bit, but the idea is that applications would be the team that works on stuff that customers see and interact with. So that’s the UI and the API. And infrastructure would be the team that customers never interact with. So any of our bank integrations, any of our SFTP transmission, any of our importers and exporters for bank specific file formats, all of that would belong to an infrastructure team, and things like the UI and the API, making sure that those things have the latest features and making sure that those products and features work well. All of that would belong with applications.

And so I'm foreseeing this split in the future, but for now with such a small engineering team, and we're constantly growing. So this might happen sooner rather than later. I definitely want to try to keep this flat reporting structure as long as possible.

[00:25:44] JM: What are some typical payment flows that get implemented with Modern Treasury?

[00:25:49] SA: So let me give you an example. So we have a health insurance company who uses us right now, and a typical workflow for them, they debit their customers who are businesses. And so let's say that a business has ten thousand dollars in insurance payments for a given month. So our insurance customer will debit their customer for the monthly insurance fee. And in this example, let's say ten thousand dollars. And so our customer initiates an ECH debit. And so that pulls ten thousand dollars from the businesses account and puts it into our customers account. And once our customer receives that ten thousand dollars, they've actually used modern treasury not only to pull that information, but to assign individual line items to that payment. So it might be a ten thousand dollar payment that they're receiving, but our customer has said, “Okay, seven dollars of this is for employee A's dental insurance, and twenty dollars of this is employee B's health insurance.” And it'll go on and on until the sum equals ten thousand. But there are individually thousands of line items within this one ten thousand dollar payment.

And so what our customer does is after they receive that and after they've confirmed it into their bank account, they have several other bank accounts. I believe it's on the order of about six or seven bank accounts that they have to then segregate this money into. So they'll take that ten thousand dollars and they'll do the individual accounting of putting health insurance in one bank account and dental insurance another account. And so a typical workflow end to end is this health insurance company takes money from its customers, businesses, takes the money, assesses a fee, does some internal accounting and auditing with Modern Treasury and then segregates the money out into individual bank accounts where it'll be held until such time as, let's say, one of these employees goes to the doctor's office and insurance needs to

reimburse that doctor. Then they'll use Modern Treasury to initiate a payment outside out of those accounts directly into the doctor's office's bank account.

So that's a typical payment flow for someone like an insurance company and it's really like I described before, a lot of our customers are taking money in on the left hand, doing some internal processing and giving it out on the right hand. And we have another company as well. It's a real estate company, a real estate marketplace, and they receive money directly from escrow. So let's say that a house closes and they receive a wire from an escrow agency, that wire will come into one of their accounts that they've connected to Modern Treasury. They'll see which house that payment is for and then they'll go look up using our system. They'll go look up which real estate agents they need to pay on their end. So for every wire that comes in, they may have to send out money. They might have to split up that payment into two payments, three payments, four payments, and all of those payments when they send it to the real estate agent, they go over different payment types. Some people want to receive ACH. Some people want to receive RTP, and others want to receive paper checks. And so they use Modern Treasury to receive a wire and then also split it up, segregate it into different payment types and send those for further along to the real estate agents. And so it's really a lot of platforms and marketplaces who are moving a lot of money in and out of their bank accounts.

[00:29:28] JM: Let's come back to what you said about the division of labor between application teams and infrastructure teams. I imagine your infrastructure team does a lot of work around monitoring and deployments. And I'd like to get a feel for just your general infrastructure stack and the kind of sane defaults you have for a given application.

[00:29:48] SA: Yeah, absolutely. And I want to preface this by saying that stability is very key to Modern Treasury. I sometimes joke that we're already the fastest treasury services provider. We don't win anything by being 10X faster, but we win a lot by being 10X more stable. And so when it comes to trading off speed for stability, I'll always choose stability. I want us to develop and ship products and features that are more stable, don't break, even if it takes a little bit longer to develop those features. And our infrastructure approach is a perfect example of that.

So we want to make sure that we have best in class infrastructure and we want to make sure that everything is stable when we deploy new features and new products to our system. So one of the primary tenets of our infrastructure is having infrastructure as code. So we use AWS. We use a specific product called AWS Cloud Formation. Essentially, AWS Cloud Formation is a YAML or JSON. We use the YAML version configuration language where you exactly specify how you want all the aspects of your infrastructure to work. And so our cloud formation configuration is checked in to its own GitHub repository. It goes through the same exact change in approval processes as it would if you were trying to change something in our codebase, in our primary codebase.

And so we approach infrastructure from this idea that, “Oh, you're deploying something new. It would be just like you if you're deploying a new product or a new piece of code.” And so in Cloud Formation we describe exactly how we want the system to work. And then using Cloud Formation, it's actually able to take that YAML definition and deploy exactly the infrastructure we want in exactly the manner that we want. And so our infrastructure specifically where we don't use any ec2 instances. This is actually surprising for a lot of folks. We are entirely using ECS Fargate containers. So when our application deploys, we're not deploying new images or new EC2 instances. We're actually just deploying new Docker containers that ECS manages the lifetime of and we're switching the load balancers over to the new Docker containers. So this really streamlines a lot of our infrastructure processes and make sure that we ship things more stably.

[00:32:24] JM: ECS does seem like the way to go for infrastructure management at this point. So you're not managing your own Kubernetes clusters.

[00:32:32] SA: No. I'm very curious what the response to this will be when this gets released, but I'm not a big fan of Kubernetes. I think it's overly complicated to not only administer but to deploy. And when there're too many moving parts in a system like Kubernetes, I get really scared. And if I don't understand large parts of the system, I tend to shy away from them. One of the things early on that I talked about with the team, and I can't remember where this is from. So I'm sorry I did not attribute the author properly. But there was this talk that was given

a while ago that became a slideshow presentation about this idea of innovation tokens. So the thinking was every company gets three innovation tokens that you get a spend on whatever technical projects you want to. And so if you're using a brand new language that nobody knows, let's say you build the whole thing in Elixir. This was a few years ago. You're spending one of your innovation tokens. And so you only have three of them. So you should spend them wisely. But if you use something boring like Java, or Ruby, or Python, you don't have to spend an innovation token and you can save that for later.

At the time we were evaluating our ECS infrastructure, I looked at Kubernetes and I thought it was vastly complicated and not that much better than what I could spin up with Cloud Formation plus ECS, and I didn't want to spend one of my innovation tokens on it. So I decided to hold on to that, not spend it and avoid Kubernetes altogether. And it’s really been great for us. I really like the infrastructure that we've built. In the future if we evaluate having to move between different cloud providers, I'd obviously revisit that. I'd revisit Kubernetes as a whole. But for now it's easy to say we're on AWS. We're not switching, and this is our infrastructure and it runs wonderfully.

[00:34:32] JM: Any other infrastructure decisions you've made that you might consider an innovation token?

[00:34:39] SA: It's interesting. It becomes hard to figure out what is an innovation token or not. One of the things we did heavily lean into is Docker as a whole. I mean our build system, when we build new versions of the codebase, what our build system is doing, we use a service called Buildkite which helps us administer our own build workers. But when Buildkite runs and we ship new versions of the codebase, we're constantly building new Docker images and ECS Fargate allows us to deploy those Docker instances very easily. But as a whole, I think Docker still feels a little new to me and perhaps to the industry. And so we've invested a lot in making a system that just prints out immutable Docker images that we can deploy independently.

So every time an engineer makes a new commit to the codebase or makes a new change, every single commit gets the full build exactly as it would in production and a Docker image is

created at the end of it. And then you can actually deploy that Docker image to one of our staging environments and test it out and make sure everything works. And this has been a great boon for our productivity. But a lot of it feels still fairly new to me. So is it a full innovation token? I'm going to say maybe it's 0.5 innovation tokens that we spent on Buildkite, Docker and Ecs.

[00:36:09] JM: What's your monitoring stack look like?

[00:36:12] SA: Datadog. I am a huge fan of Datadog. It does a great job providing us insights into the performance of our code, but also we use it as a place to send our logs to. So Datadog helps us do tracing. So every single operation or code is instrumented by the Datadog Ruby library. And so we can see where in the call stack things are taking the most time. So whether it's in a database lookup, a call to an external service or the actual code itself taking a long time, Datadog gives us a tremendous insight into how that system works. As well as when we send our logs to Datadog, it can correlate between the logs and the actual operations inside, or the actual analytics are being sent to Datadog. So if something is slow, I can go find what's being slow in Datadog and then I can pull all the logs for that event and I can actually see what the output is. So if I put some debugging output in there because I maybe had a hunch and I wanted to see what the logs are telling me, I can very easily find points in time and find all the logs for a given period. But I can also just go to a specific request and see every log associated with that request.

And so Datadog has been enormously helpful for us there and easy to administer and easy to run. We're not the type of people to host our own ELK, Elastissearch, Logstash, Kibana in production and host it and write everything ourselves. We're not ones to do that. We're much more willing to just buy a service like Datadog and have it do it all for us for a relatively cheap price.

[00:37:56] JM: Any other buy over build decisions you've made?

[00:38:00] SA: The other one is something I just mentioned about Buildkite. So instead of running our – A lot of people use Jenkins. Instead of running our own build servers or our own build software and having to orchestrate everything together, we just pay Buildkite to do that. And we have a lot of those things that come up and up again. I mean some teams have their own method of sharing secrets between each other. And we just buy one password. And the team's on one password. And for accounts that we share or passwords we need to send between each other, we just have those in one password. And so on the build versus buy spectrum, we're very much focused on buy, because it helps us move quickly and the marginal cost of buying something is worth it to us at this stage as opposed to putting an engineer on it for half their time or all of their time to figure out how to not only build and deploy these things, but also administer them going forward. I think that's a price that a lot of engineers downplay is the long-term maintenance costs. We find that it's much easier to buy in those scenarios.

[00:39:13] JM: Do you have any build over buy decisions that you've made?

[00:39:17] SA: We have. When it becomes part of the core product or when it becomes part of something that we offer directly, we tend to lean a lot more into the build decision. And one of the greatest examples of this is we have an open source project called Ledger Sync. And Ledger Sync is responsible for taking Modern Treasury's data internally and exporting it into a format that makes sense for your accounting system. And so currently it supports QuickBooks and NetSuite, but we're also adding support for other ERP and other accounting systems. And there are other accounting connectors out there. There are systems where you can glue together your system with an ERP system and have data shuffle over. But we found that the experience of using those and the product experience, really, were really subpar.

And so when it comes to delivering a direct service to our customers, we almost always go in the opposite direction and always choose the build options. So we built Ledger Sync to allow us to take data from our system and put it into accounting systems. And when we add new accounting systems, we're just adding new adapters to Ledger Sync and it's very simple and quick for us and it makes more sense when you're trying to control the project experience. Things like I described before, Buildkite, AWS, Datadog, those don't drive the direct customer

experience. And so in those cases it makes much more sense for us to buy those. But when it's in the customer flow and we're controlling how the experience works, it's almost always built.

[00:41:03] JM: Any other general engineering philosophies that you apply at Modern Treasury that you could share?

[00:41:09] SA: Absolutely. The main one that every engineer knows by heart is do it right the first time. And what I mean by that is, I alluded to this earlier, when it comes to making engineering tradeoffs, and specifically when the tradeoff is between speed and quality, I'll always choose quality. And so do it right the first time means if you're working on a feature or you're working on a product, I don't really want to impose an artificial deadline on you. I don't want to say you have to do it in this time. If you need the extra time to work on something and make it more stable, by all means, take that time. I want it to be done right the first time. Or let's say that you're seeing a product spec for the first time and there's some ambiguities, and let's say, “Okay, there're some one-to-one associations in here,” but in the future they might be one-to-many. And so we might want to think about that.

At a lot of companies I've worked out in the past, we'll always go for the quick win. We'll do what's faster. We'll do what's easier and we won't think about the long-term ramifications. And the problem with that that I've seen personally is that those long-term ramifications are not really long-term. If you have the foresight enough to think about it early on when you're writing the product, then that usually signals that in maybe three months or six months’ time it's going to happen to you. And so you'll need to change the code anyways. So do it right the first time is a mantra that we have where I encourage all our engineers and myself included to spend a little bit more time thinking about what you're building. Make sure it's more stable. But also think about the potential future implications about what you're building. Think about how the business might come to you and ask you to change something and build those things in from the start. I don't care if it takes more time. It's going to be a much more quality product at the end of the day, and that's what I really care about. If it takes more time, it takes more time.

The perfect example of this is exactly what I described, which is assuming that things are one-to-one, but later on they become one to many. I find that this is a frequent, frequent engineering challenge and business request and so many engineers kind of grit their teeth at the idea of having to do this early on and many engineers will go, “Oh, I told you so. It should have been one to many from the start. I try to give the full flexibility and freedom to our team to say, “If you know it's going to be one to many, let's do one to many now. Let's think about the current product implications of doing this, but let's make the right call today so that we're not saddled with technical debt three months from now or six months from now.

[00:43:51] JM: Can you give an example of when it's ambiguous like that?

[00:43:54] SA: Okay. I'll give you a great example of this right now. So we're implementing a feature called virtual accounts, and virtual accounts very briefly is a system that allows companies to allocate their own account numbers to their customers. So similar to how neobanks allocate account numbers for their individual customers. And our customers dictate how those accounts are allocated and who gets which accounts. And an individual virtual account early on when we were designing this product, an individual virtual account only had a single account number and a single routing number. And so it came up one of the engineer who was working on this feature asked, “Well, is it possible if one virtual account has two numbers at two different account numbers?” And so we really looked at the underlying systems and we said, “Okay, there's no technical reason a single instance of a virtual account can have multiple different account numbers.” A great use case might be, let's say, you, Jeff, have an account at a neobank and you don't want to give out your direct account number to one of your counterparties. So you generate individual account numbers and you give those out instead so you can keep your primary account number hidden or a secret. And this is a perfect use case and it was brought on by an engineer, but there's a lot of complexity or a lot of challenges that we needed to face about making it a one-to-many association instead of having one account number. You have many.

And we said, “Okay, this might be a little bit painful now, but we're going to do it. It's the right product call. It's a better product in the future. And it means three months from now, six

months from now, a year from now when this product requirement eventually comes down the line, we don't have to lift a finger. It already works and we've already thought about the potential implications of it."

[00:45:48] JM: All right, great example. Tell me a little bit about the future. Where do you see Modern Treasury expanding into and what are the difficult challenges?

[00:45:59] SA: Absolutely. I think I see Modern Treasury expanding in a few different ways. There're a few different axes that I think we can expand our business along. One of those is the number of banks that we integrate with. So currently we integrate with about I believe it's 12 or 13 different financial institutions. I want to grow that list some more. I want to integrate to potentially 20 different financial institutions and I want to add more support for different banks into our system. And so that's just a natural logical progression of our business.

I also want to expand different accounting systems. I mentioned this before. Right now we support QuickBooks and NetSuite. I want to add Xero. I want to add Workday. I want to add SAP. There're a lot of different accounting systems we can add to make our product even more valuable. And I want to expand the payment methods that we support. Right now we're primarily in the United States and we actually do have a bank in Canada. I want to expand the different payment methods that we offer whether those are payment methods internationally, and that potentially goes along with adding support for international banks. Or adding different payment methods here in the united states that we just don't have. A great example of that would be push to card, where you send money directly to an individual's debit card. That's something that we've long had on our roadmap and want to work on and want to add. And I think the more of things like this that we add, more banks, more accounting systems, more payment methods, the more valuable Modern Treasury becomes, because you'll see us as a one-stop shop for every different type of bank to bank payment and a huge amount of integrations. There'll really be no other place to turn to. I think it's very similar to Segment in that way where it's a lot of different inputs and a lot of different outputs. But at the end of the day, Segment is a wonderful translation layer for those different systems. In our case, it just happens to be different financial systems.

[00:47:59] JM: Any broader predictions for how financial services technology will evolve over time?

[00:48:07] SA: We've already start to seen financial technologies in the United States change with the introduction of RTP, and in a few years FedNow. RTP is the latest payments network to come online in the United States and I believe 40 years. And RTP is 24/7/365. When you send an RTP instruction, your recipient gets the money in their bank account within 10 seconds. It's a huge paradigm shift in how businesses think about payments primarily because RTP is 24/7/365, there's no longer a concept of a business day or a business cut off. If money can move 24/7/365, your systems now have to react in real time to payments coming in and payments going out. And so a lot of accounting and finance methodologies that are built on this idea of end of day or end of month or end of quarter become entirely meaningless in a world where money is moving around at all hours of the day. And so I think this shift is already starting to happen and it's starting to push a lot more customers into thinking about digitizing their payments as well. Check payments are still pretty big in the United States, but they're falling. Things like RTP and FedNow will help usher in more of a reliance on digital payments because consumers and businesses alike will see the benefit of an instant payment system. And we can easily see the improvements in other countries that have RTP-like systems. They are a huge improvement on the status quo and everywhere they've been introduced.

[00:49:50] JM: I noticed you haven't really mentioned cryptocurrencies. Is there anything related to cryptocurrencies that would be relevant to modern treasury or you think of that as a totally separate stack?

[00:50:01] SA: Potentially in the future we can think about uh crypto as a different payment method. I think that's the only way in which I evaluate cryptocurrencies, is it's just a different payment type. So right now in our API, it'll say ACH, or wire, or check, or RTP. In the future it could potentially say Bitcoin, or Litecoin, or Ethereum. I just see it as a different payment type. What I'm very interested in is the problems that crypto solves with regards to payment initiation, you still have the same problems that anyone else has about reconciliation, auditing

and putting all that information into your general ledger. You still have those exact same problems. So I still think Modern Treasury could help with cryptocurrencies and help solve the downstream problems. But we just haven't approached that yet on our roadmap.

[00:51:00] JM: Well, that seems like a good place to close off. Sam, thanks for coming on the show. It's a real pleasure talking to you.

[00:51:05] SA: Thanks, Jeff. Thanks for having me. This was really wonderful.

[END]