EPISODE 616

[INTRODUCTION]

[0:00:00.3] JM: When you interact with your bank, it probably feels different than when you interact with a software technology company. That's because the biggest banks in the world were started before software became such a universally important tool, and their core competency is banking, not consumer software.

Compare a company like JP Morgan to a company like Netflix. JP Morgan was started at a time when software was not at the forefront of every company. Netflix was started in the time when software was at the forefront of everybody's mind. So the development process of Netflix, the company, reflects the importance of software and it's harder to do that if your company has spent much of its lifetime out of the world of software.

But today, most banks do make some kind of consumer-facing software. Since the banks were not founded by engineers, the software development process at that typical bank does not look like the software development process at a software company like Netflix.

Monzo is a digital bank that was started with a focus on engineering. Since it was started in 2015, Monzo has always thought of itself as a software company. This gives it certain advantages over the older banks. Today's guest, Richard Dingwall is an engineer at Monzo and he joins the show to describe Monzo's software architecture, the engineering strategy and the company's migration to Kubernetes. Richard has prior experience at several different banks and financial institutions and it was really interesting to talk to him about his reflections of software engineering at previous banks and how that relates to his engineering experience at Monzo.

I want to mention before we get started, we're looking for writers. We're also looking for a videographer for Software Engineering Daily. You can find those jobs at softwareengineeringdaily.com/jobs, and if you're looking for a lower commitment way of getting involved with Software Engineering Daily, the Software Engineering Daily open source community is available at github.com/softwareengineeringdaily. You can find our mobile apps, our website as well as several other apps, and we also have those apps in the app store. So

you can work on them or you can use them if you're a power listener of Software Engineering Daily. You might find our apps useful, because they provide additional functionality beyond just the core podcast player functionality.

I hope you enjoy this episode.

[SPONSOR MESSAGE]

[0:02:48.6] JM: At Software Engineering Daily, we're always analyzing data to determine what our listeners care about, and we actually have a lot of data even though we're just a podcast. So it always reminds me that organizations with much more engineering going on have an order of magnitude more data than a podcast like Software Engineering Daily, and that's why the job of data scientist is such a good job to get.

Flatiron School is training the next generation of data scientists and helping them land jobs. Flatiron School is an outcomes-focused coding boot camp that offers transformative education in person and online. Flatiron School's data science program is a 15-week curriculum that mixes software engineering, statistical understanding and the ability to apply both skills in real-life scenarios. All of the career-changing courses include money back guarantees. If you don't get a job in six months, Flatiron School will refund your tuition and you can visit their website for details.

As a Software Engineering Daily listener, you can start learning for free at flatironschool.com/sedaily. You can get $500 of your first month of Flatiron School's online data science boot camp and you can get started with transforming your career towards data science. Go to flatironschool.com/sedaily and get $500 off your first month of their online data science course.

Thanks to Flatiron School for being a new sponsor of Software Engineering Daily.

[INTERVIEW]

[0:04:40.8] JM: All right. Richard Dingwall, you are an engineer at Monzo. Welcome to Software Engineering Daily.

[0:04:45.5] RD: Oh, thanks for having me.

[0:04:46.5] JM: You've spent a lot of your time building software for banks and these are companies like JP Morgan, Credit Suisse, a few other financial companies, and you are now working at Monzo, which is a newer bank that was just started in the last, I think – What? 4 or 5 years super recent?

[0:05:06.5] RD: Actually, it's 2015.

[0:05:08.2] JM: 2015. Okay. There's been this explosion of new fintech businesses, whether we're talking about new banks or new payment gateways or new payment systems, why is that? What caused the explosion of new businesses in the payment space?

[0:05:23.8] RD: I think for us it was in 2014 when the PRA, so that's the Prudential Regulation Authority, that's one of the regularity bodies here in the UK that oversees banks. When they introduced a new streamlined banking license application process, which is called the mobilization route, and that enables new banks to sort of soft launch in a kind of private stuff only test mode before fully ramping up to accept new customers, and this was done to basically make it easier for new companies and to encourage competition, because historically there haven't been many new banks for a long time. So this was a time when Monzo and a number of other challenger banks appeared.

[0:06:01.4] JM: How much of this was due to AWS and other infrastructure as a service process? Was it strictly this regulatory benefit or did it have anything to do with the lower cost of infrastructure and all the additional services that have been built in the last – Oh, let's call it 10 years, this boom of new infrastructure and developer tools that are much cheaper than things in the past.

[0:06:26.9] RD: So I don't think cloud providers like AWS were a requirement for starting banks like Monzo, but I think there were two really big benefits for companies like us. One was allowing us to do iterative development with our IT infrastructure, kind of in the same way that we do iterative development with our software. So we don't know what the final product is going

to look like before we start. It's a journey and we're going to change it many, many times and try a lot of different things before we kind of arrive at an optimal solution. If we needed to be kind of provisioning physical hardware changes in our own data center every time our infrastructure changed, the cost would have been much higher and it would have slowed us down a lot. So that was the first benefit.

The second one is really about talent. So any person with a laptop can sign up for free AWS account and start building quite sophisticated things. So we have much broader access to really smart engineers who are already familiar with the internals of our platform, which is really cool.

[0:07:25.4] JM: You're saying that the talent of the average engineer was raised, and I guess the broadness of the domain that some random engineer if familiar with, rather than being familiar with the Microsoft Stack or the LAMP Stack. Now you have a preponderance of engineers who are familiar with the AWS Stack.

[0:07:47.8] RD: That's right. We call them sort of children of the cloud. There are young engineers here at Monzo who've never worked in businesses which have had physical servers on premise or anything. They've always worked in Amazon or Google's cloud.

[0:08:03.8] JM: Starting a bank, we have lots of banks. Why did Monzo need to start a new one?

[0:08:09.2] RD: So I guess it was really the opportunity. I think there had been quite a lot of sort of stagnation and sort of feature development and products in banking. As I said before, there hadn't been much new competition in the market and all the banks were sort of broadly the same and, really if you kind of logged in to your bank account, your online banking in sort of 2007 or 2017, there really wasn't apart from a sort of a CSS redesign, there really wasn't a lot of new kind of features or development happening there. So I think we really saw that we could do things much better, we could basically take – We want to basically give power to customers, like the sort of the power that you'd get from other online platforms like Gmail for managing your email, for example. Your bank, your online banking should be as powerful as that.

[0:08:59.9] JM: If your anybody in the audience like me, you know that banks make money in a variety of ways. What are the main ways that banks make money?

[0:09:11.2] RD: I guess the classic kind of way for a bank to make money is called net interest margin, which is basically the spread between interest by the bank paid by customers on loans minus the interest paid out by the bank to customers for deposits, which had a credit balance.

So at Monzo we make – And as well, I guess, in the UK we have free current account banking, but it's not really free because there are a lot of penalty charges which can be quite easy to fall into and some of them maybe are not very well aligned with the customer's needs.

So at Monzo we make money by interchange, which are fees paid by the merchant to accept card payments, interest we earn at the Bank of England on our deposits, marketplace partnerships. So, for example, if we identify that you could save some money by switching your energy provider, because we can see your energy provider bills and we get some commission for that if you choose to proceed with that then we would, both bank and the customer, would benefit in the same way. We also have overdrafts, although we kept these rates very low so you can't end up racking up huge bills and lending. But in the long term we're trying to ensure that our incentives are aligned with the customer so we don't profit though things like penalty charges when you're in financial trouble.

[0:10:37.5] JM: So there are some of these business sectors where you can end up in a place where your incentives are adversarial to the customer who has their money deposited in that bank. What are some of those relationships and which ones have you selectively avoided?

[0:10:56.0] RD: I guess the one that kind of pops to mind is overdraft charges. I think something like 25% of banking, current customers in the UK, are in persistent overdraft. So it's really people that I guess kind of less well-off, who maybe have less sort of financial security. So if you're in a credit balance, but you don't pay anything for having an overdraft, that if you are in a negative balance and you're struggling to kind of make ends meet, those people are the people who are paying the most and they're kind of fueling, I guess, the profit to the highest degree, which we don't think that's really acceptable. So we do have some overdraft charges to kind of cover the – We do take financial risk there, which has a cost, but we kept it very low so there's

no risk of these sort of ever growing charges. You know exactly what the maximum charge would be very clearly.

[0:11:53.1] JM: When I think of a traditional bank, I think of an organization with a lot of organizational overhead that may not be necessary that perhaps Monzo has gotten an opportunity to rethink, and you get this organizational overhead because perhaps the company builds its software infrastructure 20, 25 years ago and so you have certain processes that you develop in the company that are crutches for the technology that's 25 or 30 years ago. This is natural and that's true of every organization. What are the advantages that you have? Since you get to start from scratch with modern technology and you get to rethink everything from scratch, what are the things that you have an advantage over where perhaps you have something automated, you don't need a human in the loop.

[0:12:46.6] RD: I think for us, I mean, one of our kind of goals is to be what would a bank account look like? I guess we're kind of inspired by companies like WhatsApp. I heard WhatsApp have something like 300 million users and 55 engineers, which is this incredible scale. So our goal is to be like what would a bank look like if it has a billion customers and is able to scale like a company like WhatsApp? So I guess right from the beginning we've been able to kind of design our products and our processes in ways that scale not linearly, but I guess we can edge large numbers of new customers without needing to – Maybe we need to scale up our AWS build to increase the number of nodes running in a platform, but we don't need to higher, thousands more employees or build new branches, things like that.

[0:13:39.8] JM: And in WhatsApp case, the problem domain was scoped to something that was easier for everybody in the company to understand, which was advantageous for them and their incentives were very straightforward. I believe it was subscription-based business and they're just like, "Let's just build a really good, really performant messaging system and we'll keep our team as lean as we can, because if that's all we're trying to do then we should really focus on that competency and just build really good relationships between the engineers on the team and do it better than everybody else instead of sprawling and doing all kinds of things that may or may not be necessary.

So can you just focus? Is there some specific area of technological competency? Because on the case of WhatsApp, they focus on the messaging layer and so that boils down to a really performant erlang system, which runs at a super low cost, and that's a pretty interesting downstream result of a high level business strategy. So how does that affect the mindset within Monzo from the business level? How does that boil down into engineering decisions?

[0:14:54.9] RD: I guess it translates. If you want to have a bank with a billion customers and with quite a low number of stuff, you need to basically design processes and products and things that require kind of no manual input or entry. For example, with a lot of banks, a lot of traditional banks that offer like a mobile product or an online banking or phone banking and you can do certain operations you can do through your web browser. There's another class of operations which you can only do over the phone, and then maybe an even higher class of operations that you can only do in person. So that doesn't scale, so we need to ensure that all operations you can do everything through the mobile app. In many cases, customers are able to solve their own problems directly.

We actually shipped a feature – Well, we're in the process of shipping a feature at the moment where, say, if you go to a shop and you buy something, or usually it would more happen more likely to be an ATM when you might get charged twice, and one of the charges will eventually drop off after a few days. But if you are charged in error or, for example, the merchant terminal says decline but the bank says it's approved and it takes some of your money, you can actually reverse those charges yourself through the app. I'm not aware of any other bank before kind of shipping a feature like that where customers can actually basically cancel the transaction themselves and get the money back for it obviously within certain safely limits and so on to prevent kind of abuse. I think that would kind of inform the product decisions which would then inform the technological decisions.

But in that case, it's really just, I guess, the sort of being a platform that is robust and doesn't have outages, we don't have scheduled outages, for example, we don't have scheduled maintenance. We are on everything, kind of active-active, multi-master.

[0:16:53.5] JM: That feature of enabling people to cancel transactions, that's a feature that you would only be able to implement if you had sophisticated fraud detections software, and I think

the mentality around fraud detection has become democratized. So if you go 20 years back, a company like PayPal, they could build a core competency in fraud detection and that could be their competitive advantage as a payments company.

Today, the knowledge of how to do fraud detection is much more wide-spread. To what degree have you found fraud detection to be a solved problem and to what degree has that been actually something where you have had to build more of a competency and innovate more?

[0:17:42.2] RD: It's really interesting question. I think for longtime, Monzo was – I guess there are different types of fraud you can see on a bank account. So first party fraud, which is where a customer uses their own account to do something nefarious. Then third-party fraud, which is someone maybe getting access to someone's bank account and stealing all their money or something. I think basically we do all that kind of fraud detection in-house. We have a mixture of kind of discreet rows and also some kind of machine learning models. That's kind of been a journey for us basically as we've grown as we sort of shipped new products, new ways to pay things, and it's one of those cases where you kind of always one step ahead, or trying to say one step ahead as people are kind of probing your system for sort of vulnerabilities and flaws and things you can do. But we do find that fraudsters typically do kind of follow similar patterns to each other. In general, I think we've done pretty well, but it's certainly been a case, I guess, similar to what was – I think, with PayPal, they were losing like huge amounts of money I think right at the beginning, but it was almost like a development cost. To build a really world-class fraud system, you need to have fraud so you can identify what it looks like and how to recognize it and how to put systems in place to block it.

[SPONSOR MESSAGE]

[0:19:09.6] JM: At Software Engineering Daily, we have a web app, we have an iOS app, and android app and a backend that serves all of these frontends. Our code has a lot of surface area and we need visibility into problems that occur across all of these different surfaces. When a user's mobile app crashes while playing a podcast or reading an article, Airbrake alerts us in real time and gives us the diagnostics that let us identify and fix the problem in minutes instead of hours.

Check out airbrake.io/sedaily to start monitoring your apps free for 30 days. Set up takes only a few minutes. There's no complicated configuration needed, and Airbrake integrates with all of your communication tools, from Slack, to GitHub, to GIRA and it enhances your current workflow rather than disrupting it. You can try out Airbrake today at airbrake.io/sedaily. If you want to monitor and get visibility into the problems that maybe occurring across your application, check out Airbrake at airbrake.io/sedaily.

Thank you to Airbrake.

[INTERVIEW CONTINUED]

[0:20:29.8] JM: That's the exact same conversation I had with Stripe. I did a show about Stripe's Radar product, which is a fraud detection product they have and a lot of the subjectivity to the fraud detection software that their building is how much fraud do we let in, and it's like a knob. You can turn the knob to accept more fraud and then you get more training data and you also get fewer false-positives, people that are falsely flagged as being fraudulent, but the consequence is you probably lose more money on people who actually are defrauding you.

When you look at those products, like a Stripe, Radar or a Smite is another company I'm friends with. They do not exactly fraud detection, but a more general service of detecting bad actors. Do you think this is something that's going to be completely commoditized, where if I'm a company that has any sort of transaction process, I can get fraud detection as a service, because otherwise that's such a big development cost for anybody that'd doing transactions.

[0:21:38.6] RD: I think it depends what your interfaces are that people could commit fraud through. So for a lot of business, just accepting card payments online, I would be very confident using Stripe for that. I think when there are certain – I think you need to look at your own product and see how can people exploit your own product. Can you make payments through your product? Something like that. Can you abuse refunds or things like that? Can you ship products before kind of transactions are fully confirmed? What's your kind of [inaudible 0:22:12.2] stuff?

I think for a bank that's more complicated because we have a lot more interfaces by which you can make payments and each one of them has its own sort of fraud attack paths, which people can do, which you need to kind of discover. Then there are other products out there as well for sort of sharing information about like basically when you do kind of detect fraudsters to share that information with other companies. So you might jointly blacklist them. I think we use mostly kind of police and government agencies for that sort of thing.

[0:22:43.2] JM: Let's get into engineering of Monzo. Describe the life cycle of a transaction. So when I deposit a check into an ATM or I make some kind of payment, walk me through some of the different systems that a transaction is going to touch.

[0:23:02.3] RD: Let's go with making approaches with your card. So say you walk into a shop to buy a sandwich or something, so the checkout staff will enter the amount into the terminal. They'll ask you to insert your card. Let's say – If it's a chip transaction or a mag strip transaction or a contact list, the parties involved are still the same. So the merchant terminal would basically – If it's a chip transaction, there would be a conversation between the card and terminal. That protocol is called EVM. The merchant terminal would then send that data to – Could be the merchant systems or it could be a payment gateway. That would be the company who would provide the payment terminal to the merchant. They would perform a bunch of security risk checks and sort of fraud checks to say maybe this is like a card we've blacklisted before. They would then forward that on to an acquiring bank. So this is a bank which doesn't have retail customers. Its customers are the merchants. The acquiring bank would be connected to the card payment scheme, like Mastercard, for example.

So the acquiring bank would then run a series of more checks. They would then forward that message on to the card network. The card network would run even more checks. They would then send that to the issuing bank, which would be Monzo, for example, that we would then receive that request as like a sort of a request response message. We would then either just make an authorization decision whether to decline or to approve it, and if we approve it, then we would update a ledger and then we would return a response back to Mastercard and then back to – They would return the response back to the acquirer, which goes back to the – And eventually all the way back to the merchant terminal. After that happens then we would kick off a bunch of asynchronous processes for things like transaction creation and a sort of fee

enrichment and things like that, and all of that needs to happen in, I think, 9 seconds, which kind of sounds like quite astronomically high amount of time, but actually we find in – I think that's a requirement that we find in practice and it really happen in about one second otherwise we see transactions get canceled.

[0:25:15.0] JM: So the payments infrastructure, if that's your highest throughput volume, how bursty is that? Is it a very steady load of traffic throughout the day or do you have a lot of spikes? Is scalability, up and down scalability a big concern for you?

[0:25:37.6] RD: I think in terms of – It's basically, I guess, kind of daily – We have one big spike in the morning, so when everyone goes to work and a lot of people buy coffees or things like that, then it goes a bit quiet until lunchtime and then there's another spike again at the end of the day for sort of rush hour going home. The last sort of Fridays of the month when everyone gets paid, that's always a big day. Apart from that, it's basically just sort of seasonal human behavior, basically.

[0:26:06.9] JM: The thing that drew my attention to Monzo in the first place was the openness in talking about the infrastructure, and part of that was the discussion of migration to Kubernetes. So you migrated from Mesos and Marathon, to Kubernetes. Could you tell me about that process, and I think throughout that conversation we can probably explore in a little more detail what your infrastructure actually looks like.

[0:26:36.9] RD: So I think we were quite an early adopter of Kubernetes. I think it was like June or July of 2016. So this was the time before we head out for banking license. So basically before getting a banking license is a slow process. I think it took us a couple of years. Before we were allowed to issue our own cards and basically host a bank ourselves, we ran a prepaid program where basically we used an agency and we had our own branded prepaid Mastercard cards where customers could load money and then the app works pretty much the same as the current account app works. So you get sort of full experience and we were able to do iteration and development.

But at that time, the system of record for our customers balancers was this third-party card processor system who they had a license before while we were still working on ours. So at this

time we realized we probably wouldn't have another opportunity to re-platform before launching the current account on our own infrastructure. So on the migration, we basically cloned the bank. We were running two Monzos, if you like. We used Terraform to bring up a new production environment running Kubernetes alongside the old one. Then I think a few stateful services, like Cassandra, which we used. That's our main database, and message queues, like NSQ and stuff that was shared between the environments. Then we just switched traffic over. I think we've actually done that more than once since when we've done things like Kubernetes upgrades as well.

[0:28:03.5] JM: We'll get into the Kubernetes stuff in a little more detail. Small side note, Cassandra, as your main database, what's the reasoning behind using Cassandra?

[0:28:14.9] RD: Cassandra, it's a distributed key-value store with no master nodes. All the nodes are created equal, which means there's no sort of failover and it runs this sort of ring topology, where basically keys get written into tables. They get hashed and then written into a table on a node according to whereabouts in this ring it hashes to. You can tune the level of consistency. You can say, "I want to write these values into this table, but the write should only succeed if definitely it gets written to a minimum number of nodes." So you don't have your data – Your data exists on multiple nodes at once. It works really well for us. It has incredible write performance and read performance. I think pretty much all of our kind of read and write operations earns sort of sub 10 milliseconds and it's really – I think it's one of the things that kind of really powers Monzo.

[0:29:08.8] JM: We've done other shows about Cassandra. So if people want to know more about how and why to use a masterless replicated key-value store, they can check those out. But could you maybe contrast it with other options? For example, what would be the penalty of using a MySQL database instead of Cassandra, or trying to use MongoDB instead of Cassandra?

[0:29:34.9] RD: I don't have too much experience with MongoDB, but with a traditional relational database, basically you would need to sort of use transactions to write into that database with kind of consistency and to ensure that your write isn't going to conflict with other

people's writes and you're not going to read data, which is in the middle of being written by someone else.

Cassandra doesn't provide – Doesn't have kind of acid transactions like that. So you kind of need to do that yourself. So we at Monzo, we actually use etcd for that, which is another distributed key-value store which runs – We run it in memory. Can you hear that clapping?

[0:30:10.8] JM: They're clapping for etcd.

[0:30:12.1] RD: Yeah.

[0:30:12.5] JM: Etcd, solving transaction [inaudible 0:30:14.2], solving Kubernetes consistency deserves applause.

[0:30:18.5] RD: Yes. So at Monzo – Yeah, we use etcd to basically provide that locking. So we would have a service which should be a process running on a machine somewhere, which is writing to Cassandra. Before we do that, we would acquire a lock in etcd, which is basically a distributed key-value store which runs in memory. So one service would insert a key into etcd with a lock ID, which would probably be something like the customer ID or something, basically the scope that we want to lock on. Then other services if someone was trying to do a concurrent read or current-current write and they wanted consistent view of the data, they would also try to acquire that same key in etcd, and if the key already exists, they would wait until it is deleted, because that would indicate that another service has acquired that lock. Then, yeah, when that lock is available, then the other service – So the services are basically serialized. It basically allows you to do kind of serialized access to things, serialized access to anything just in your service. So that's how we kind of lock around Cassandra.

[0:31:28.1] JM: That makes sense, although it sounds like a lot of work to go through in order to ensure locked database transactions. You got to imagine, there are some cloud-hosted services that give you all of these stuff out of the box. I don't know if it's Spanner or some other cloud service. Do you have any – I don't know if you have any more contrasting detail for why Cassandra over the other – Perhaps the hosted solutions, or is it an element of four years ago maybe you just simply did not have this offering available as a service on any cloud provider?

[0:32:06.4] RD: I think this is one piece of infrastructure that we did want to run ourselves. My understanding actually before I joined, these sort of decisions were made, but my understanding is that, yeah, maybe we could use a database like Spanner or something, but then maybe that would tie us to a particular cloud, and I think we have a lot of operational experience running Cassandra. So I think we were kind of more confident to pursue that.

[0:32:28.7] JM: Cool. With Cassandra, do you have to do an export of the data in order to do analytical processing on it? If you want to run aggressive machine learning jobs on it, you don't want to be hammering the Cassandra database itself, right?

[0:32:45.6] RD: No, and being a key-value store is actually quite difficult to do sort of table scan type queries. Basically in Cassandra you can only look things up by direct ID lookups or by ID range kind of scanning, which is useful for things like time series when the ID might be – Or one of the partition key might be a time stamp. But at Monzo we use something called liked a fire hose architecture, where basically all of our services emit events, they publish messages to a message queue when things happen.

So, for example, one feature that we have you can freeze your card through app. So if you want to just disable your card, maybe you've misplaced it and you're not sure. You're worried about someone finding it up and making fraudulent transactions. So you can just freeze it while you look for it. So that would involve a write to – Or acquiring an etcd lock with something like the card ID as the lock key writing, updating a row in Cassandra to mark the card as frozen, which would then cause any transactions to be declined. Then publishing a card .frozen event to a message queue.

Two happens with that. One, that gets used for triggering asynchronous behavior in our platform. Other services could then listen that. An example of that would be if your card is added to Apple Pay or Google Pay wallet, we have to synchronize the state of the virtual card in the wallet with the physical card. So that would trigger that sort of asynchronous behavior so you're not waiting for that as you freeze your card and it happens afterwards. Then the second place that those events go is to BigQuery, to Google BigQuery, where we run a lot of kind of analytics. All of our kind of, I guess, that step that's – So basically you can those for really

anything, sort of observing sort of customer behavior, observing how people use the app, things which are happening, so bulk aggregate, reporting and analysis, building funnels to see where people are using certain features and drop off roads and all sorts of stuff. So BigQuery is super useful for us. We use it a lot.

[0:34:54.9] JM: The fire hose architecture that you described, I think that would overlap with some people's definition of event sourcing or CQRS, this family of ideas where a change to your data model starts with a message to a distributed message queue, which in your case is Kafka. Kafka has the pub/sub queue and other data sources can read from that queue and update their own models of the world whenever they get the chance so that you have a distributed append only queue of the event history across your platform and you have materialized views that are more responsive to particular query patterns, whether you're updating Cassandra for your transactionality, or you're updating Elastricsearch for your searching and updating your BigQuery cluster for analytics. You have different patterns for accessing the data and updating things. So you use Kafka for that. Why Kafka? Why do you use Kafka over other messaging systems?

[0:36:10.2] RD: As a bank, I think it's kind of driven by business requirements, really. As a bank we need to make certain guarantees that things actually happen. So an example, we can't commit to making an important directive at payment if you have like a bill for your mortgage, like your monthly mortgage payment or something. Yeah, we're going to make that payment for you on the third of each month or something. Then we actually try it, it times out and that's it, or maybe we – I don't know, a message got lost somewhere or something. I think Kafka provides really good guarantees that a message has been published. It also has this kind of training pool replication factor semantic to Cassandra so you can say, "Publish this and it should only succeed if it gets published to this many nodes." So at least kind of N-copies of your message on different nodes so you can tolerate the loss of a certain number of nodes, it's masterless as well. Again, there's no single point of failure and there's no kind of failure process that needs kind of testing.

One thing about it, it's strictly ordered. So many operations in our platform don't require strict ordering. If you and I both try to make a payment to pay our friends at the same time and one happens before the other, that doesn't really matter. That certain things where we do kind of

constrain, we do kind of serialize things. So in our ledger services for example, that does require strict ordering. So Kafka is really good for that.

[SPONSOR MESSAGE]

[0:37:45.8] JM: Citus Data can scale your PostgreS database horizontally. For many of you, your PostgreS database is the heart of your application. You chose PostgreS because you trust it. After all, PostgreS is battle tested, trustworthy database software, but are you spending more and more time dealing with scalability issues? Citus distributes your data and your queries across multiple nodes. Are your queries getting slow? Citus can parallelize your SQL queries across multiple nodes dramatically speeding them up and giving you much lower latency.

Are you worried about hitting the limits of single node PostgreS and not being able to grow your app or having to spend your time on database infrastructure instead of creating new features for you application? Available as open source as a database as a service and as enterprise software, Citus makes it simple to shard PostgreS. Go to citusdata.com/sedaily to learn more about how Citus transforms PostgreS into a distributed database. That's citusdata.com/sedaily, citusdata.com/sedaily.

Get back the time that you're spending on database operations. Companies like Algolia, Prosperworks and Cisco are all using Citus so they no longer have to worry about scaling their database. Try it yourself at citusdata.com/sedaily. That's citusdata.com/sedaily. Thank you to Citus Data for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:39:30.4] JM: It’s useful as a medium for this fan out messaging to different materialized views of your overall data model, but you’re not using Kafka. If I understand it correctly, you’re not using to have services communicate with each other. So you have micro-services architecture where different services have necessity to call other services and you get these distributed traces of big chains of services that are calling each other, because each service is scoped to some specific thing. Maybe it’s calculating a tax that you have to get, or it’s grabbing an update to the rate of – I don’t know, financial something, but you’ve got lots of different

services that are specific to some narrow domain. You've got narrowly scoped micro-services. Tell me about how the services communicate with one another? In contrast to this pub/sub notion of messaging, I mean, services also have a notion of messaging each other. But as I understand, unrelated to the notion of publishing to a message queue for updates to your durable stores.

[0:40:46.9] RD: Yes. I think it's good to look at basically how the services talk to each other. So at Monzo we use micro-service architecture. Instead of running maybe a handful of very large kind of monolithic service processes, which dispatched things using function calls within one process, we compose systems out of a lot of smaller components, which all run as individual processes and then talk to each other.

So we have two main types of communication between services, synchronous requests, which we call remote procedure call or RBCs. This is for things where we want to wait when we need to know the result if they succeed or fail. For example, making an ATM withdrawal should wait while we validate your pin and it should decline if it's unsuccessful.

The other type of communication is asynchronous messaging. So as I said, we use those when a response is not immediately required. For something, when we want something to happen, but it doesn't need to happen immediately and we're not going to wait around for it. So an example of this would be like when you make a purchase, we send you a push notification to your phone to tell you out that, but we wouldn't want to put that push notification on the critical path of the actual transaction initiated from the payment terminal. We want to put that on an asynchronous message queue then it will process that afterwards. For a synchronous RBC transport, we use HTTP. For our asynchronous transport, we use Kafka.

[0:42:12.5] JM: What are the messages that are being sent over HTTP? What format are they in? Because you do JSON over HTTP. You could have GRPC over HTTP. So what are you using for the messaging medium?

[0:42:29.3] RD: So at the moment we just aim for simplicity at the moment. So they're just JSON over HTTP. We kind of chose HTTP as our RBC transport, it's because all programming languages have HTTP service and HTTP clients and can talk to each other. If we've written

some custom, some very clever custom RBC client library with all sorts of bells and whistles, we would need to implement that in every programming language before we could use it. So the vast majority of our services are written in Go, but some programming languages are better equipped for solving certain problems. So that's why we kind of chose HTTP, and we just use HTTP and JSON in the moment. I think it's likely we would switch to HTTP 2 quite soon and possibly GRPC, but for now, yeah, just plain JSON works pretty well for us.

[0:43:18.1] JM: Why do people switch to GRPC? What are the constraints that they might hit with JSON that would cause them to switch to GRPC as their messaging protocol?

[0:43:30.7] RD: I'm not super familiar with GRPC, but I guess from my kind of limited understanding, the serialization is faster and I think can do code generation. Basically, generating all of your sort of message types in whichever language and also service definitions so you can have the sort of DSL for describing your services and messages and then generate clients in different languages that can talk to each other.

[0:43:57.9] JM: Right. So if your organization started to sprawl and it got really big, maybe you want to have more type safety or constraint on who can send what message in what format, and you probably also can tighten up any latency sensitivity because you have the better serialization layer. Anyway, there're other shows where people can look into that.

In this inter-service communication, you use a service mesh. So I've done several shows about the service mesh in the abstract but have not talked to people about service mesh in application at any particular company that is using service mesh as a customer. I've only talked to people who are building service mesh technology. Why is service mesh useful to you?

[0:44:47.4] RD: Yeah. I guess maybe first I'll kind of explain how it actually works in kind of basic terms and then sort of the benefits it provides. So we use LinkerD as a service mesh in between services. What that actually means – So all of our services are deployed in Kubernetes and on nodes. We deploy LinkerD as a daemon set on – So it runs on every single node and it runs as an HTTP server. So all of our services, if they want to make an HTTP request to another service, they make a request, an HTTP request to local host with the name of the destination service in the URL path just as a string. LinkerD then handles that request. It knows where all of

our service processes are actually running, so on which node and which port by talking to the Kubernetes API. Rewrites the URL with the IP and the port of the node containing the destination service and forwards it on.

So all of our services just think they're making requests to local hosts, but actually the request gets rerouted to the destination service, which it might be on the same machine or it might be on a different machine. If the request fails due to like a transient failure like a timeout or something, LinkerD will actually wait and then retry the request up to a certain kind of retry budget and up to a certain number of attempt.

So basically you can write code that just makes a very, very basic HTTP request with absolutely no kind of bells and whistles to local host and you'll automatically get service discovery retries, circuit breakers, all of that kind of good stuff that gets uniformly applied across any HTTP request between our services.

[0:46:36.6] JM: In contrast to a world where I'm not using a service mesh, I would have to write custom routing logic and retry handling logic and circuit breaking logic within my service potentially, which would mean you've got programmers who should just be developing services also writing this code that is in some sense a commodity across the entire infrastructure.

[0:47:03.2] RD: Exactly, yeah, and you would need to kind of – You also maybe need to tune those parameters at run time. There's quite a lot of configuration to maintain. So having LinkerD kind of automatically do it between requests just makes it a lot easier for us.

[0:47:17.9] JM: So we jumped to other infrastructure topics, but this migration, you dropped 75% of the infrastructure cost overtime with this migration to Kubernetes and the process was, as you said, you spun up a clone of your infrastructure. For a while, you had an increase in infrastructure cost, but overtime you were able to realize, "Okay. This is stable," and gradually move more and more of your traffic off of your old infrastructure to Kubernetes. As I said, you dropped, I think, 75% of the infrastructure cost, and that's amazing. But is infrastructure cost a big percentage of your overhead at Monzo? It seems like the majority of the overhead is going to be engineering man hours or woman hours make the case that this is a worthwhile migration despite the engineering cost.

[0:48:16.7] RD: I think that was a nice benefit and it certainly – Yeah, it was very interesting, and I think the cost saving I understand was mostly down just to Kubernetes kind of been backing algorithm for running containers on a node and kind of how it allocates, how you can kind of allocate resources to them.

Yeah, the kind of infrastructure costs are quite big, but certainly much smaller than, I think, our stuff. Our stuff costs are much higher. I think that's sort of our main cost at the moment.

[0:48:46.4] JM: Right. There is probably a massive increase in developer productivity because of that migration.

[0:48:53.4] RD: Yeah. I think Kubernetes is really great. Everyone can ship things whenever they like. Everything's very hand-on. I think it just gives tremendous amount of kind of power for sort of engineers to own code all the way through to production rather than sort of handing things over the teams to actually operate.

[0:49:15.5] JM: I know we're up against time. I just had a few other questions to conclude with. So we did all these shows recently about crytocurrencies and the cryptocurrency ecosystem. Do you think at all about how the traditional banking industry like you're operating within in Monzo will integrate with the crytocurrency world?

[0:49:38.4] RD: It's a really good question. I think at the moment we don't really have any plans. I think it's at the moment I guess there are few sort of applications where you could use it. We could offer Bitcoin trading to our customers. We know some customers would absolutely, absolutely love that and be really over the moon if that happened but we also appreciate it. So it's still as successful as kind of Bitcoin is, as popular as it is. It's still a very small minority of our customers asking for products like that.

I guess our ledger, it's a centralized non-cryptographic ledger. I thin for banks, I think it makes more sense to basically centralize at the moment in terms of customer, our own kind of customer's movements of money inside our own bank. Yeah, I think at the moment we don't really have any plans, but I guess we're well-positioned if something comes up.

[0:50:34.0] JM: What are the other kinds of aspirational technologies that you could eventually layer into a core banking system? If you took a bank and you gave it the aspirations of – A company like Amazon or Facebook or Google, you can do these exercise where you extrapolate the business 10 years into the future and be like, "These are probably the things that they're aspiring to right now, or some of the things that they're aspiring to and they're futuristic and they could be exciting." But if you're inside of a bank today, or maybe a financial exchange, you may not be privy to the same level of excitement and ambition, but it seems like this is probably something that you do have as a competitive advantage within Monzo. You probably have that aspiration, that ability to see applications of crytocurrencies, or augmented reality, or who knows into how it could actually be useful for a bank. What are some of the futuristic aspirational technologies that get thrown around at 5 PM on a Friday when everybody's kind of frazzled and just getting excited about what could be in store for the future?

[0:51:50.3] RD: There's actually not too many that come to mind. I think we're quite cautious. We do kind of – I think in some ways we absolutely dive in and go crazy every time there's a new iOS feature that we can take advantage of, operating system feature that we can take advantage of or a new android feature that we can do something cool with.

In terms of other kind of technologies, I think we kind of play around with a lot of stuff, but it's really just about things to make money work better for people, also just making our own systems more resilient. Yeah, I don't think there's really any – I can't think of any kind of aspirational kind of flashy technologies that we're super bullish on.

[0:52:36.8] JM: They're clapping for you once again.

[0:52:39.8] RD: [inaudible 0:52:40.0].

[0:52:41.5] JM: Okay. Richard, that's a valid answer. In these interviews, I actually got emails from people recently who were like, "You did too many shows on Ethereum and the craziness that could be built within an application level blockchain," and there's a whole lot to be done with just money. You just make money work. It has a lot of positive impact on the world.

[0:53:03.4] RD: There's actually is one thing. Our public API, which we put on the backburner a little bit as we focused on just building up our current account just to get that working really well for our customers. I think there's really some exciting things we can shift through that. So sort of merchant integrations, things like basically replacing – You might carry cards with like stamps on them, sort of loyalty cards, things like that. Imagine if a merchant, if you buy something and the merchant can actually enrich, can actually add things into your bank statement to provide more details of your purchase, or like your warranty information or like shortcuts to an exchange or something, or a refund, things like that. I think that's a really interesting area which we are looking to spend a lot more time on the next few months.

[0:53:48.1] JM: Yeah, I'm sure that one metric of measuring how much ground there still is to be covered in traditional banking is how thick my wallet still is. I've got bus passes, subway passes, loyalty cards, credit cards. Then I've got this smartphone and I'm like, "Why do I have all these other things?"

[0:54:05.7] RD: Yeah. I mean, hopefully we can tokenize them into the secure element of your phone and get those – Yeah, get your wallet smaller.

[0:54:13.9] JM: Richard, thanks for coming on Software Engineering Daily. It's been really great talking.

[0:54:16.3] RD: Yeah, thank you very much.

[END OF INTERVIEW]

[0:54:20.8] JM: At Software Engineering Daily, we have user data coming in from so many sources; mobile apps, podcast players, our website, and it's all to provide you, our listener, with the best possible experience. To do that we need to answer key questions, like what content our listeners enjoy? What causes listeners to log out, or unsubscribe, or to share a podcast episode with their friends if they liked it? To answer these questions, we want to be able to use a variety of analytics tools, such as mixed panel, Google Analytics and Optimizely. If you have ever built a software product that has gone for any length of time, eventually you have to start answering questions around analytics and you start to realize there are a lot of analytics tools.

Segment allows us to gather customer data from anywhere and send that data to any analytics tool. It's the ultimate in analytics middleware. Segment is the customer data infrastructure that has saved us from writing duplicate code across all of the different platforms that we want to analyze.

Software Engineering Daily listeners can try Segment free for 90 days by entering SEDAILY into the “How Did You Hear About Us?” box at signup. If you don't have much customer data to analyze, Segment also has a free developer edition, but if you're looking to fully track and utilize all the customer data across your properties to make important customer-first decisions, definitely take advantage of this 90-day free trial exclusively for Software Engineering Daily listeners. If you're using cloud apps such as MailChimp, Marketo, Intercom, AppNexus, Zendesk, you can integrate with all of these different tools and centralize your customer data in one place with Segment.

To get that free 90-day trial, sign up for Segment at segment.com and enter SEDAILY in the “How Did You Hear About Us?” box during signup. Thanks again to Segment for sponsoring Software Engineering Daily and for producing a product that we needed.

,

[END]