EPISODE 626

[INTRODUCTION]

[0:00:00.3] JM: Nubank was started in 2013 with a credit card that was controlled through a mobile app. At the time, Nubank was the first service in Brazil that allowed customers to do banking without going to a physical bank branch. Since then, Nubank has expanded into additional financial services and currently has 850 employees in Brazil.

Edward Wible is a cofounder and CTO of Nubank and in this episode he discusses his work growing Nubank from a small team of less than 10 people into a company with almost 1,000 people. We've covered two other banks in the past few weeks, Monzo and N26. In terms of software engineering and product management, Nubank is similar to Monzo and N26.

One characteristic that stood out was Nubank's use of Clojure, which is a functional programming language built on the JVM. A question that came up during the show, “What is the line between a fin tech company and a bank?” We hope to explore this question more in future shows about the intersection of money and software.

[SPONSOR MESSAGE]

[0:01:19.0] JM: You listen to this podcast to raise your skills. You’re getting exposure to new technologies and becoming a better engineer because of it. Your job should reward you for being a constant learner, and Hired helps you find your dream job. Hired makes finding a new job easy. On Hired, companies request interviews from software engineers with upfront offers of salary and equity so that you don't waste your time with a company that is not going to value your time. Hired makes finding a job efficient and they work with more than 6,000 companies, from startups to large public companies.

Go to hired.com/sedaily and get $600 free if you find a job through Hired. Normally, you get $300 for finding a job through Hired, but if you use our link, hired.com/sedaily, you get $600 plus you’re supporting SE Daily. To get that $600 signing bonus upon finding a job, go to hired.com/sedaily.

Hired saves you time and it helps you find the job of your dreams. It's completely free, and also if you're not looking for a job but you know someone who is, you can refer them to Hired and get a $1,337 bonus. You can go to hired.com/sedaily and click “Refer a Friend”.

Thanks to hired for sponsoring Software Engineering Daily.

[INTERVIEW]

[0:03:01.6] JM: Edward Wible is the CTO and cofounder of Nubank. Edward, welcome to Software Engineering Daily.

[0:03:06.4] WB: Thank you very much. I’m happy to be here.

[0:03:08.5] JM: It's great to have you. We've recently done some shows about some financial technology companies in Europe and the United States. How does the banking system in Latin America differ from that of the United States or the UK?

[0:03:22.0] WB: So there's a few differences that are important. One is market factor, which is concentration. So in Brazil, 90% plus of all the assets in the country are held by the top five banks. So that's a high degree of concentration. The US has a low degree of concentration, not quite that concentrated, but the long tail is a lot longer in the US. I think there's something like 5,000 banks.

So, Brazil, it’s really about the big banks, and kind of when we created Nubank, the goal was really to take a slice off the big banks because that was where the market was at. I think another factor related to the market is the Central Bank and the centralized banking infrastructure for the country. On that one, Brazil is actually really strong, especially compared to the United States with real time grow settlement, kind of peer-to-peer transfers with the hub and spoke model throughout the financial system. And a lot of that, from what I understand, I haven’t lived in Brazil for very long, but I think that's a legacy of inflation. Managing a hyperinflation area economy requires computers, and so that was one of the reasons that banks kind of

implemented a lot of technology early on. Some of that technology now getting in their way in the form of legacy COBOL mainframes and such.

[0:04:38.3] JM: Can go a little bit deeper on that? Why does a hyperinflation area economy lead to the development of computer systems?

[0:04:47.8] WB: So I think this is a net settlement environment whereby you are comfortable building up counterparty risk until the end of the month and kind of net settling, right? So we won't communicate between banks for every single tiny transaction, right? Were just going to net it out and we'll do one at the end of the month. So we can each kind of keep our own bookkeeping systems and there's not a high service area integration. There's not a kind of high cardinality of interactions in the network.

In an inflationary economy where things are changing every day, A, just managing your policies and your strategies and kind of keeping the bank running as you expect it to. It's important to be doing that quickly and automation is really the only way to keep up with that. If you're in a low trust environment where you don't want to be relying on other banks in the system being sound or trustworthy, right? Settling in real time for the gross amount rather than waiting for the month end, especially if numbers are going to kind of keep drifting on you throughout the course of the month. It makes people more comfortable. So those are good use cases for automation and banking.

[0:05:53.2] JM: So here you've outlined a stark difference between the banking system in Brazil and perhaps other areas of the world. Does that impact you from an engineering perspective in building Nubank?

[0:06:10.5] WB: I think it does. So there's a couple ways that it impacts us. First is that what we find is the system is strong. So the fundamental architecture, the backbone of the financial system is really strong. So we're actually quite happy whenever we can interface and integrate directly with systems kind of managed and controlled by the central bank of Brazil. So that's interesting, because that's basically an endgame for us as a financial institution, at least as a Brazilian financial institution, which is what we are today. We have's expanded internationally yet, right?

The Central Bank or Basel is the central monopoly. So if you get a good high-quality integration with something central, you're kind of done, right? I think the other aspect of that where I think Brazil shows a bit of its immaturity as a market is that there's not a really well-developed partner ecosystem here.

So what we find is that integrating with others to do things like build payments, or to sell for instance insurance through our channel. Those sorts of integrations that people in Europe, like N26, or Monzo, they have options. I find that in Brazil, we don't really have reliable options. So from an engineering perspective, we end up doing a lot from first principles and a lot of stuff in-house that you might actually consider maybe a little bit less than core in some other markets.

[0:07:38.5] JM: It's funny that you mentioned N26 and Monzo. Those are two companies, both of which I'm interviewing, or I have interviewed Monzo and I'm interviewing N26 pretty soon. So it will interesting to see the contrast between the different systems that these new banks have built.

I've also talked to some other companies transfer-wise, Stripe, I'm curious what you think is the definition of a bank versus a fin tech company. Is it just a matter of where you start? Can you go from being a fin tech company to a bank or go for being a bank to a fin tech company? Does that delineation actually matter?

[0:08:20.3] WB: That's a great question, So I'll try to answer that. I'm not sure I fully know the answer, but I'll give it a shot. So what I see from N26 and Monzo in particular, but there's some others. I see that a simple bank in the US was another example. Most of these challenger banks – And I think those are challenger banks. Not necessarily fin techs, but they start on the debit side, right?

So from our perspective, we haven't launched debit yet. It's something we're working on, but the domain model for debit is a lot simpler, right? Everything from the settlement cycle, to the way chargeback works, to installments, which are very Brazil-specific concept where people buy at the point of sale with "interest free installments". So there's is a lot of things that we dealt with by going credit first, credit cards first, that are a little bit simpler in the debit side. But other things

are a lot harder on the debit side, right? So you have to build a brand and build trust with customers to give you their money and to give you their life savings, right? Whereas, on the credit side, we were giving customers credit lines, right? In a scary scenario, that's free money that you never have to pay back, right? That's a charge off or diabolical in the industry.

But on the other hand, it's something safe to try and you can kind of pay your bill and you realize that it's free and it all just works. So I think that's one difference between where we start. I could be wrong, but I think that our strategies are quite similar. I think, in every case, we're trying to be the financial institution, and that means do our own KYC, know your customer and accept the liability for that. Fight money laundering, interface directly with regulators, produce regulatory reports, be the system of record, right? Actually control the financial relationship with our customers, and ultimately use our own balance sheet, right?

So I think that a lot of financial businesses that exist today such as interchange on a debit card or a credit card where transaction fees are really the name of the game. I think on a long enough time horizon and with enough competition, a lot of that goes to zero. The essence of what it is to be a bank will be the only thing that remains, right? Which is you provide services for free. You get data that you can use to correlate with credit outcomes. Then you can use that data to underwrite people for loans, right? And that's really the business model. You take deposits in order to have a funding source for your loans and you manage the duration of your deposits versus the duration of your loan book. That's really kind of – That's how it's been for thousands of years, and I think that that's a very robust business model that will continue to be attractive.

I find that in the US, a lot of the fin techs try to not be a bank for reasons that may be very legitimate, such as we don't want to comply with Basel, right? We don't want to have capital requirements. We don't want to have to be regulated. These are all kind of reasonable things. It's very expensive to be regulated. It's very expensive to have Basel III capital requirements, but it really depends and if you believe you can have a robust business model if you don't really control those elements, right? Like if you're a service provider or a market maker and you kind of are matching up peer-to-peer lending, I think history has – Especially recent history has shown us a bit that that's a fragile model. So our goal is to be really be the financial institution and be a tech company that happens to be a financial institution, right?

I think that fin tech is a bit of a broader term also whereby you have lots of different segments of the value chain, right? You may have fin techs that provide know your customer and identity fraud management as a very narrow niche, and that's a fin tech, but that's not a bank. So to come back to answering your question, I think that might be my definition, is the fin tech spectrum is very broad and there's a lot of niches, but the bank is really about the balance sheet and the level of trust and the level of criticality in an economy and the relationship with the regulatory bodies.

[0:12:27.8] JM: So we knew bank was founded, the differentiating factor of the company was this credit card. You got a credit card that was completely controlled through a mobile app, and that was in 2013. I think that was enough of a product to just start to build out a customer base and start to have an understanding of what people wanted out of a financial company. What was required to build that initial product, the credit card that was just controlled through a mobile app?

[0:12:56.5] WB: Sure. So, initially, the idea that the basic competitive idea that we had was that credit card really could be disconnected from branch networks and from the physical experience of going to a bank. There's nothing physical about a credit card. In addition, we thought that credit card could be re-architected to work as a real time mechanism as opposed to a monthly batch or something like that where until you get that PDF in your email, you don't really know where you stand, right?

There's online banking and there's varying degrees of latency to know where you stand, but actually accruing everything in real time and making all the mechanics of credit card work in real time turned out to be far more challenging than we had anticipated in the beginning.

But thinking back to 2013, the first critical milestone that we had was integration with MasterCard, right? At the time, Visa wasn't very interested in talking to startups. I remember them not being responsive to us. I think that situation has probably changed quite a lot over the last five years, but MasterCard was. So we needed to get a contract and then an implementation, a chip project for the plastic chip and pin card and all those things done with MasterCard.

I wasn't as closely involved on that side, but what I understand was that we were one of the fastest projects to kind of go to completion in MasterCard's history. That was really the long pole in the tent for us, because at the time, in 2013, there was no regulatory approval needed to be an issuer of credit cards. In 2014, I think, May, that regulatory window closed. So any institution that was fully operational, operating kind of with customers at that time, would be allowed to continue operating until formally regulated, right? Only today are we getting our first audits from the Central Bank about the –It's called an IP, payments institution license, right?

So if we would've had to shut down the company and wait until we got a license and then start operating, that would've been, I guess, the death of the company at that point. So we worked very hard to launch a working credit card and that had everything to do with the MasterCard project before the regulatory deadline. So at the time we gave literally equity incentives to the engineering team to ship something that worked by April of 2014, which we did with a he team of about six at the time, I think.

[0:15:28.6] JM: That's sounds like a really high pressure engineering situation.

[0:15:32.3] WB: I think it was also a high ambiguity situation. I don't mean ambiguity in the sense of how do we find product market fit, kind of lean startup type thing, because we knew we were building a credit card. Everybody knows what that is. So we kind of skipped the sacrificial architectures and stuff that where's we're going to iterate quickly until we figure out product that's going to work, right?

We knew which product will work. We just didn't know how to build it and we didn't know how to build it necessarily in a way that would scale. So learning the domain model was really probably the hardest part. We ended up starting with a micro services architecture, but we got the bounded context wrong in a few different ways.

One very painful way was modeling money as an integer number of cents only to find later that there are such things as fractional cents, things that come from Facebook ads, for example. So that was a bummer. I still remember for the moment when the team realized that we have made that mistake.

Another one was sizing certain services, right? We made a service called accounts and it did accounts, right? And then subsequently, we had to split out billing, line items, credit card accounts, probably three other services out of that. So we set out to build a micro services architecture and ended up with an accidental monolith. So I think some of those things were we weren't sure of the scaling, the throughput needs and really how the thing was going to break down. That was challenging.

One aspect of getting into the market that quickly, so actually doing an authorizer for MasterCard, and again I'm not so sure about Visa these days, but building a MasterCard authorizer requires a physical data center and a least line and some specialized hardware that gets shipped from the United States, something that getting that through Brazilian customs alone could have blown up our timeline.

So we ended up doing an integration with a third-party that already had a working authorizer, and this was, in many ways, a deal with the devil that also saved the company, right? Something that I don't regret to this day, but a lot of the problems we had subsequently in getting the thing to scale and kind of managing the transition from a third-party being the system of record and us being more like a monitor, kind of showing the information that the third party claims is true to us kind of slicing that salami to the point that we are the system of record and we are accountable and we stand behind our own numbers, right? That was something that was very, very challenging and was only really realistic for us to do after we implemented the authorizer infrastructure, which was 18 months or two years later.

[SPONSOR MESSAGE]

[0:18:19.2] JM: Support for Software Engineering Daily comes from Wunder Capital, an easy way to invest in large scale solar energy projects across the United States. With Wunder Capital, you can earn up to 7.5% annually while helping to financing renewable energy projects. Get started with Wunder Capital at wundercapital.com/sedaily. That's W-U-N-D-E-R-C-A-P-I-T-A-L.com/sedaily.

A few years ago, we did an episode of Software Engineering Daily where I interviewed somebody from Wunder Capital about how the software infrastructure works there and as well as why people would invest in solar-based projects. So if you want to learn more about Wunder Capital, you can certainly listen to that episode.

Since 2015, individuals have invested tens of millions of dollars using Wunder Capital's solar investment platform, and alongside individual investors, Wunder Capital also works with financial institutions. There is a prominent Wall Street hedge fund that has invested over $100 million with Wunder Capital. So if you're interested in investing in businesses going solar, go to wundercapital.com/sedaily. That's W-U-N-D-E-R, wunder with a U, wundercapital.com/sedaily.

[INTERVIEW CONTINUED]

[0:19:55.7] JM: Today, you're a bank. You are a significant large institution. Give me a high level view of your engineering stack today.

[0:20:05.4] WB: Sure. So I think the in the Brazilian market from a regulatory perspective, we have to be a little bit careful about saying we are bank. I think we are a payments institution. That's a license that we have. I think that being a bank and being a payments institution are slightly different in terms of what we are allowed to do how we do it, but I think the customer doesn't really care because the experience and the relationship with us really doesn't change. Whether we're using third-party balance sheets, third-party banks, interfacing, or whether we're actually kind of building a [inaudible 0:20:38.8] and interfacing with our own [inaudible 0:20:40.0], which is a which is a form of a banking license that we've also applied for and been granted but haven't implemented yet.

So, with that caveat, the stack is actually really consistent with how it was in 2013, 2014. It hasn't changed a lot. I don't know if that's something we should be proud of or worried about, but what we see today is about 160 engineers in the company, about 170 micro services. Most, if not all of the micro services, are running on closure. So pretty simple closure web apps, Jetty, Pedestal web framework. Most of them are communicating with Finagle on the HTTP side or

with Kafka for async messaging and they're all running in AWS. So that's a really simple architecture.

I think the hard part isn't deciding to do JVM. It's a bank build on the JVM. The hard part for us was very much understanding the domain as we started as a team with very little or frankly no domain experience, especially not in Brazil, and we needed to figure out which 170 micro services to build and how would they interact in a way that would be idempotent and scalable. Really kind of optimize for not creating bugs that are hard to catch bugs where you've made financial mistakes that impact customers that you find out later.

Surely, you can always kind of reverse things out, right? But there's a level of trust that people expect from financial services. I'm not saying they always get it, but when they don't get it, we kind of skip the patient customer phase and go right to I'm irritated, right? Like you did the math wrong. So that's something that I think has been challenging for us to do from first principles, but also really powerful, right? We've rewritten core banking from scratch, partially because we were naïve and we didn't have any idea how big that monster was to really rewrite it, right? But also partly because we didn't fit into the kind of mental model of the big vendors in the space, right? Talking to T system or kind of one of these core banking vendors. They wanted more than all of the money that we had to do an initial implementation of the technology, right?

So it was kind of necessity and also kind of just in a reckless naïveté. But in the end I think we emerge as something that's kind of unique and fairly interesting, and I get the sense that Monzo at least is also pursuing a similar path, right? A lot of gas from first principles, a lot of technical principles, right? We're a functional programming shop not in terms of really esoteric concepts, but just in terms of the basic immutability, idempotence, declarative format, small functions, pure functions, right? Make sure everything is easy to test. Those sorts of things that service really effective guiding principles even for 160 engineers that are trying to write closure services in the same way.

So I have been very impressed with the way that architecture is scaled. To this day, we don't really see big reasons to change. I think there's some exciting new technologies coming on the pipe. I see sometimes people are talking about Rust, and that's cool. I think sometimes you have the strong type system communities versus dynamic type system community types of

conflicts happening within Nubank, but really overall just happy with the way that people are able to get on the same page and we're able to write even technology that's actually quite a large system at this point without having any single piece of it that is difficult to understand or impossible to kind of rip out and rewrite and replace. So I think that's a testament to micro services as opposed to the mainframe as a model for maintainability and kind of evolving the stack over time.

[0:24:35.2] JM: There's a lot there in what you just said, and I want to start at the language level, when we're talking about some aspects of the stack that you've standardized. At the language level, you've got Clojure, which I think that's like a functional language that runs on the JVM, right? It's like functional Java.

[0:24:54.8] WB: It's a lisp as well. So a lot of parentheses.

[0:24:57.6] JM: Right. Now, I think I've talked to somebody at Jane Street Capital, which is a trading company and they use – Oh! What language do they use? OCaml. But I think the reason they use it is kind of similar to what you're implying, which is that when you're dealing with money, you want strong guarantees around certain aspects of the programming model. What guarantees do you get from a functional language? If that conversation extends to aspects of the JVM that you can take advantage of that are also virtuous given your domain, the domain of money, the domain banking, the domain of needing to understand transactionality and rollback things potentially. Tell me about the language choice of Clojure.

[0:25:52.0] JM: Yes. So I think Clojure is an interesting choice because it's very simple. So it's very foreign to people. So it looks like an alien, right? So when people come into Nubank, we're typically not hiring people that have Clojure experience. If there are people that have Clojure experience, we're happy to hire them, but we can't count on that because it's a relatively niche technology. So hoping for a large, passionate experienced community that will kind of come in to Nubank with everything they need to be seniors is not a realistic strategy. Although it has been a good strategy for us on the self-selection, kind of positive self-selection side, right?

What we find is that people look at the prospect of learning Clojure over a weekend in order to do a technical exercise as something that they might attempt. Even if they don't fully pull it off,

we find that those are often you courageous people that are sometimes dissatisfied with the status quo, dissatisfied with the traditional way of building systems and they tend to be really good cultural fits for us. So we've been happy with that.

In terms of the way we build systems. I think having functions, a bunch of small functions that compose and having really strong common libraries that tie all of our service together has really helped us, and Clojure allows you to do that because the cost of using a function from some other place is really low, right? Whereas the cost of kind of using a big object-oriented abstraction and kind of this tower of dependency injection and all that good stuff, I think it can be significantly higher, right?

So for everything from stubbing and mocking and having a bunch of really sharp knives, but really small functions that kind of do small things that you want them to do and they do them really well, right? So it lowers the barrier to code reuse and it simply helps us to find clarity in our systems around what is a side affecting function and why, right? So we adopt the convention of ending any function that causes side effects with an exclamation point, right? And this is not necessarily a Clojure thing, but it's something we do at Nubank. And if you see a lot of functions with exclamation points, has a great smell to react to and try to understand, because on good service, on a normal service, what we see is 90% plus of all the lines of code and all the complexity is in the form of pure functions. And then the side effects, like producing a message, receiving an HTTP request, writing to a database, all that good stuff, tends to happen at the at the edges, right? It happens just at the borders and changing which database, changing which message queue, changing how you get HTTP, how you send HTTP, none of that really is that difficult, and most of those side affecting things happen in code that's really well tested from common libraries.

So I think trying to get a lot of people to write Java in that way and kind of training them and then chasing down deviations from the style guide and stuff, I think it'd be really hard. But I think Clojure, that kind of the smell and the friction when you're not doing it in a functional way is built into the language. So everyone kind of notices it. In a current scale, everyone kind of notices deviations from common patterns, because there's a lot of copy and paste from another service that's really helpful.

So I think I credit Clojure with a lot of maintaining consistency without kind of centralizing everything, finding a way to still actually maintain a relatively homogenous architecture and have that scale. That's been really helpful.

One last comment on Clojure. I think there are multiple different kind of sub-communities of Clojure around how you feel about types, right? And one of the more recent introductions to the Clojure language has been Spec. But before that, there was something called prismatic schema or plumatic schema and it's basically a way of – It's not types and kind of the formal sense, but it's a way of describing the data as it flows through the system. This function expects to take inputs that have these characteristics and produce outputs that have these characteristics, right?

The most common case, programming at Nubank, and it also really helps with the readability of Clojure code, which can be a bit cryptic. So from very early days, we've used types and enforced types specifically at the level of integration tests, single service integration tests are really the work course for Nubank in terms of assessing correctness of flows, and those tests check all types throughout all the calls from kind of outside the system, in through Kafka, out through HTTP, into the database and back out, all that stuff.

So it's a way of doing it typed, similar to a typed approach but with a dynamic language, and I think we've gotten kind of the best of both worlds for our model, for our context. I think that saying, "Which one is better?" is something that we probably want to leave out of scope for this conversation.

[0:30:57.6] JM: You gave a few descriptions earlier about some other standardizations around services. So you said services used to Finagle to communicate with each other. If I recall, Finagle is a service proxy. It's the thing that eventually led to the LinkerD service proxy project and it allows you to have some sense of standardization around services when it comes to things like load-balancing, and I think service discovery, and circuit breaking, and things like that. You also mentioned that you use Kafka for asynchronous communication.

Explain some more aspects of the service standardization model and some best practices around your services as they stand by themselves and as they communicate with each other.

[0:31:52.8] WB: Sure. So Finagle was a project that came out of Twitter and it really distills out a lot of distributed systems, kind of black magic that Twitter figured out is like failure accrual budgets, exponential and jittered back-office, things where back pressure and circuit breaking and stuff is really important to make sure that 170 services don't trample one another, don't cause a chain reaction and cascading failure.

So we see huge value in leveraging everything that Twitter learned versus trying to kind of reinvent that wheel and learn all those same lessons as we put more and more pressure into our systems as we scale.

So Finagle – We used to use Finagle on both ports, HTTP and Kafka. HTTP being our main kind of synchronous I/O. Kafka being our main asynchronous I/O. We also used a circuit breaking that was also integrated with rights to the database. We use a database called Datomic, which is rarely used, but is a really good conceptual fit with things like Kafka and Clojure, because it maintains an immutable log abstraction kind of on the inside similar to the Kafka log, but it also provides entity semantics and a powerful query language via data log. So you kind of get all facets of your data without ever losing any data. There's no update in place. There is no you change this field and you don't ever know what the prior value was, right? It behaves more like Git.

So I think that's an example of something where you're writing to the database and the database is fallen over and then you want to make sure that you back propagate that failure so that you stop consuming new messages from Kafka and stop doing useless work and logging out exceptions and basically just slow the whole system down until things can recover. So that's a good example of where Finagle helps us.

I also find Finagle to be an interesting example, because Finagle is written Scala. So we do a little bit of Clojure Scala interop, which is not pleasant, but it's a testament to the JVM being kind of like the lingua franca, because we've leveraged really awesome, really mature library ecosystems. We kind of say it's the JVM, right? So it scales and you can use it for financial services. So a lot of the original concerns around Clojure being a bleeding edge language, "Is

that safe?" There is a little bit of hand waving around, "You know, it's the JVM, right? Everybody uses the JVM."

So I think we've seen good benefits from using Clojure, which I've heard it referred to kind of having something very pure and beautiful and amazing, kind of ancient technology from a sophisticated race of aliens, then kind of welded to the chassis of the Death Star, right? I think that somehow that it works and it works really well for us.

Kafka is what we like to do whenever real-time semantics are not important. I've heard about people doing ways, doing things that make Kafka kind of feel real-time work where it kind of block an incoming request until you get kind of Kafka coming back with something. The commander pattern from Bobby Calderwood was a really cool video about that. But we don't really do that. We generally use Kafka in situations where async okay and we typically classify queues into ones where we expect no lag. We expect this to be up to speed and we expected to perform really fast versus things were we may fill up a queue with the big batch job that we kind of split line by line, one message per line, or one message per chunk, or whatever, and then lag is different. So the monitoring around that is tuned to that use case.

But we find that Kafka, we tend not to do so much kind of rewinding and kind of replaying of the past, even though you can do that with Kafka via offset management. But we just tend to use it as a really good distributed system that is partitioned, and as long as you're setting partitioned keys, you can kind of have a guaranteed total ordering. On a per customer basis, that actually allows you to avoid a lot of complexity inside the service, because Kafka deals with it.

So I think we could use Kafka way more. We're starting to use Kafka Streams and starting to explore the possibilities of what we can do with it, but it's early days. So today I think we can underuse Kafka given the potential that I see.

[0:36:28.4] JM: What would you use Kafka Streams for? And describe What Kafka Streams allow you to do.

[0:36:32.3] WB: So one use case for us – So we have an accumulate only sort of data model, right? And a customer's balance or the customer's available limit is a cumulative function of

everything that has ever happened in that customer's life, right? You need to know the credit limit changes, the limit increases, limit decreases. You need to know every transaction, every payment. The cumulative sum over all those things is your available limit on your credit card. So it sounds simple, but it tends to behave and perform badly if you don't have caching. But luckily it's a great fit for caching, right? We also do sharding, where we'll put kind of segments of that on a timed basis into cold storage.

So I think Kafka Streams is a good model for things like push caches, where you're accumulating state and you know that you can accumulate that state safely because everything that's going to happen for that customer, for example, is going to have the same partition ID, which means it's going to end up at the same node and it's going to end up getting there in the right order, right.

So what that allows you to do is make distributed caches, right? Instead of having a Reddis associated with every service, you could actually just accumulate that state in memory, RocksDB, whatever, per instance and let Kafka decide when instances go down, when instances cycle, when instances scale up, kind of outsource that the Kafka to figure out how to rewire the partitions and to make sure that consistency is maintained and the quorum stays in sync. So I think caching is the short answer to your question, is a really good use case for Kafka Streams.

[SPONSOR MESSAGE]

[0:38:19.2] JM: In today's fast-paced world, you have to be able to build the skills that you need when you need them. With Pluralsight's learning platform, you can level up your skills in cutting edge technology, like machine learning, cloud infrastructure, mobile development, dev ops and blockchain. Find out where your skills stand with Pluralsight IQ and then jump into expert-led courses organized into curated learning paths.

Pluralsight is a personalized learning experience that helps you keep pace. So get ahead by visiting pluralsight.com/sedaily for a free 10-day trial. If you're a leading team, discover how your organization can move faster with plans for enterprises. Pluralsight has helped thousands of organizations innovate, including Adobe, AT&T, VMWare and Tableau.

Go to pluralsight.com/sedaily to get a free 10-day trial and dive into the platform. When you sign up, you also would get 50% off of your first month, and if you want to commit, you can get $50 off an annual subscription. Get access to all three, the 10-day free trial, 50% off of your first month and $50 off a yearly subscription at pluralsight.com/sedaily.

Thank you to Pluralsight for being a new sponsor of Software Engineering Daily, and to check it out while supporting Software Engineering Daily, go to pluralsight.com/sedaily.

[INTERVIEW]

[0:39:58.4] JM: Let's take a step back. So you're the CTO of Nubank and how many engineers do you have at this point and how big has the engineering team gone?

[0:40:06.7] WB: So the engineering team is about 160. I think 161 as of yesterday.

[0:40:12.0] JM: 161. So you've gone from a place where you had six engineers to 161. Tell me about the process of scaling that org structure. What were the points along that road from 6 to 160 where you had to do some reframing of how the engineering team worked?

[0:40:32.8] WB: Sure. I was actually a cofounder of Nubank. So I was the first employee in a sense. So we went from kind of me to 160. The employee-base in total is about a thousand at this point, about 750 customer service and engineering kind of being the largest non-customer service function in the company. So a lot of the kind of scaling learnings for product teams, we encountered first in the engineering team, and then other functions that Nubank tend to kind of adopt similar models.

But the first one we hit when we were about 15 people was it was just a mess. Like nobody knew what other people were working on. Everything was kind of tangled up. So we decided at that point to split into two different teams. I think at the time it was one team focused on customers and another team focused on back-office. Something like that.

Then from there, this concept of a squad was formed. I think later on we kind of started using similar terminology to the Spotify model. Although that the Genesis was really just looking at the problems that we saw and trying to solve those as opposed to trying to connect cargo cult somebody else's org model into our company.

I think in 2016 we actually started rolling out the kind of small teams. Amazon does the two pizza teams. I think we did something similar. It's probably 4 to 6 engineers ideally on each team, and there's about 20, 25 teams today that have engineering out of probably 40 squads within the company. So today we do have some squads that don't have engineers.

But really the idea to combine engineering with other functions, like business analysts, or product managers, even customer service was a decision we made early on, and that actually works really well to co-locate customer service with engineers when things are small. I think when customer service scales up, if you kind of scale with headcount at the point of theirs four engineers and 40 customer service reps, that's not really a team anymore, right?

So I think there's kind of a beautiful moment where it works and the feedback loops are really tight and you learn a lot. But we kind of outgrew that and now it's more a representative model, right? So there are representatives that kind of connect with the squad, but it's not like we're all in the same squad with customer service these days.

I think the other big change that we made over time was evolving our mentality around management. I think in the beginning we had a bit of an immature thought process about it, right? It's like that's overhead. It's pure overhead. Do we need it? We don't. When Google had 300 engineers, they eliminated managers. Then of course they brought them back and there's a lot of nuance to how to support people and what management really is that I think nobody in the company really understood.

So we undervalued that. As a result, we undervalued and kind of didn't model behaviors and didn't model the management track as an attractive career path. So one of the problems I have today is incentivizing that track and scouting talent for the management track. Whereas kind of most engineers at Nubank think of themselves as proceeding along the technical track of the Y-career.

But today, with 160 engineers, we have basically three layers, right? There's need and there's a layer of engineering managers or directors, kind of the middle management layer. Then there's the squads, and each squad has a technical lead. Sometimes the technical lead role is split into two, like a technical mentor who's the senior technical person and a technical manager who's the senior kind of people management lead.

So sometimes it's two different roles. Sometimes it's kind of can both combined in one person often out of necessity, right? Not enough seniority on a team to actually have two people sharing or doing two different parts of the role. Then there's the kind of the terminal units is 4 to 5 to 6 engineers on those teams. So that's an org structure that has gotten us here and it's going to get us quite a long ways further, and I think in some sense it's absolutely traditional, right? After a certain scale, hierarchies are the way to keep things organized. So not having started there, I think I've kind of ending up there, similar to most of the other big tech companies that we know about.

[0:44:59.9] JM: This is what I heard from a number of different companies that I've interviewed and companies that I've just heard on other podcasts, or in audiobooks, or whatever, where they say they used to think hierarchy was a bad thing and n traditional org structures are a bad thing. Then they realized, "No. Actually, companies just do that because it's a solved problem and you probably don't need to do holacracy or some crazy, completely flat org structure. It's just not necessary. You can just kind of borrow from the past with these org structures.

[0:45:36.3] WB: I think the thing that I had to really realize was the role of managers in a servant leadership sense, right? Removing roadblocks, providing kind of career guidance, right? Doing one on one's. Understanding the value of one on ones. Doing one on ones in a way that isn't a status report, right? Getting to more awkward, kind of more meaningful conversations around what people want, right?

I think a lot of those things is when management's not done well. People confuse it with trying to control other people and tell other people what to do. We find that Nubank is that managers don't really do that very often, right? They're more there to observe, recognize patterns, remove

roadblocks and basically kind of pick up on the signs of when teams are unhealthy and ask questions until you kind of get to some insight there.

So I think once engineers at Nubank realize that that's what management is, even people that similar to myself were not very excited about in the beginning I think today are quite supportive. So I think we've really come full circle on that and I'm proud of how far people were able to kind of interpret and assimilate new information and change their points of view over time.

[0:46:49.3] JM: So as far as that, the one on ones and the soft skill style relationships with people that are working for you when you're a manager, how do you develop that and how do you find the right modality of interacting with an employee? Because I'm sure you you've had this experience where you can't form to close of a bond with an engineer that's working for you. You don't want to cross from the role of manager versus employee relationship to like friend where we're hanging out style relationship, or at least there's a certain danger. You endanger the hierarchy when you do that, which might sound unfamiliar to people who haven't been in management before, but for people who have been in management, they know the problems that can emerge from that.

So what's the right line between that hierarchical relationship and the friendship, the problematic friendship when you're talking about the manager-employee relationship?

[0:47:50.6] WB: Right. I find that when this is a problem, it's because the power dynamics are really the elephant in the room and people don't you just introduce the elephant, right? So I find this is common with new managers who were previously on the technical path or this is their first time managing people, and sometimes those people were former peers and they don't want to own the power that they have, right?

So talking about that and introducing the elephant in the room I think is really the best way. I think that coupling a formal acknowledgment that there are power dynamics, right? Like I'm going to be deciding things like promotions, compensation, hiring, firing. Although hiring in Nubank often we do in a more centralized way, firing typically in the team. But that's evolving. I think just making the roles clear and say, "Yeah, I'm going to be playing this role, and this is

what the company is asking of me and I'm going to play that role," but coupling that with empathy.

So one of the reasons I like technical managers at Nubank to continue to code for at least part of their time is, in my experience, the first thing you forget when you stop coding is that humiliating kind of things are two weeks late. You worked all weekend and you realized that you were solving the wrong problem. The stuff that's really humbling, you forget. What you remember is, "I remember that one time when I would have shipped this problem, I would've done this in two days. What's taking so long?"

So that sort of impatience I think stems from a lack of empathy. It's okay to be unsatisfied with where we are, but you don't want to misdiagnose the problem and kind of come up with the wrong root cause for that, right?

So I think combining really clarity around the role in the power dynamic with empathy for the software development process and what it really feels like to be an engineer including what it feels like today, because it changes over time. I find that that's usually enough to not make the boss this kind of distant adversarial figure, but also not eliminate the sense of camaraderie and kind of we're all on this together in many ways friendship, right?

But, yeah, are the engineers that work at Nubank my best friends? Do I hang out with them every weekend? Not really, right. There is a certain distance, which I think stems from mutual respect hopefully on both sides.

[0:50:15.9] JM: I know we're running out of time here, and I wanted to ask a few more questions about the future. So what I think is pretty interesting about building a fin tech company today is if you just build a foundation of a nice financial product that has a lot of usage, you've got a lot of tailwinds that you can take advantage of in the near future. So you take a company like Nubank, which is either you have a credit card product, or maybe you're a bank. You can't describe yourself as a bank, but you're something like a bank. You've got high transaction volume. You've got users that love you. Or you take a company like Stripe where they've got a large volume of users that are just using it for simple payment features. You got these tailwinds,

like distributed ledger technology, and machine learning that could improve how you're doing loan assessments or various risk assessments.

So you have these tailwinds that really just improve your economics for whatever foundational financial technological system you have built today. Assume you agree with that premise, what are the tailwinds that you're most excited about that you think you're going to get the most value out of?

[0:51:31.7] WB: So that's a great question. One of the things that I'm really excited about is probably as far removed from the kind of distributed ledger blockchain conversation as it could get, which is I'm excited about building a high trust relationship with our customers, right? Because if they trust us to do their accounting for them and to help them get through their financial life and they respect us and they find that they can rely on us for that, and if we can do that in a way that's pleasant for them and low friction, which is something that I think existing financial institutions, they do have the confidence of customers. They also have a lot of the frustration and ire of customers, and that's kind of a cautionary tale.

But if you can do it in a way where it also feels a bit like WhatsApp, or Twitter, or Facebook, something that's really easy to use, then I think the customer will trust you to do more things for them. So then the oldest strategy in financial services, like the cross sell thing, you really make that a viable strategy, right?

I think you also kind of future proof your business against the things yet to come, right? So if there are going to be supranational kind of blockchain, cryptocurrency syndicates that are important for the future of economic life on the planet, someone is going to to connect those things to people walking around on the street in San Paulo speaking Portuguese, and I doubt that it's going to be a startup in Singapore.

So I think that's the key, is to figure out if you are the last mile, if you're terminal unit, in which case we want to be as Brazil specific as we can. We want to be as Portuguese specific as we can. We want to understand everything about the customers here and we want to understand everything about the regulator here, right? So we want to build customized reports for the

Central bank. Those are very different incentives from like an Uber kind of galloping across the globe at breakneck speed.

So I find that that's pretty interesting. So I like being the terminal unit. Of course, replace Brazil with some other country for to GO International. I tend to think we would have to rewrite 40% to 50% of all the services we've written if we were to go to another country, because that's probably the proportion of our technology that is Brazil specific. That's also one of the reasons that we haven't open source a lot of technology at Nubank because we've been really standing on the shoulders of giants and solving problems that are fairly specific to us, which is a way of considering whether we've prioritized things appropriately.

So I think that's something that I don't know how to predict what exactly will be the killer app for blockchain and kind of Bitcoin and all that stuff. But I think that owning a customer relationship and kind of not doing anything to violate that trust is a pretty good place to start, right?

If I'm wrong and kind of decentralization really is the trend, then nobody needs Nubank anymore, right? Like we are a central kind of trust authority. So either that's useful. I tend to think it is. Or it's not, in which case we'll get different jobs, I guess.

[0:54:38.8] JM: Indeed. Well, Edward. It's been really fun talking to you. I appreciate you coming on the show and I'm really excited by Nubank. I'm excited to see where you go next. Congratulations on the success and the scale that you've reached at this point.

[0:54:51.5] WB: Thank you very much. It actually made me think of one other concept about the future that may be worth touching on just in closing.

[0:54:56.5] JM: Sure. Go for it.

[0:54:57.8] WB: So I think that the credit underwriting piece is a big one, right?

[0:55:00.9] JM: Yeah.

[0:55:01.6] WB: So that's really interesting, because it's proprietary information on some level, right? We get to learn which algorithms we want to use to predict customer outcomes, and that has really good economies of scale, right? More customers, more data, more track record, more history. All these things are disadvantages for startups versus the big banks. But if you can kind of cross the chasm and get to the other side of having some scale and having some track record, that's kind of the second horizon. The first Horizon is probably user experience and just getting rid of some of the nonsense and the friction, right? Which is pretty simple, but it's also something that everybody's going to replicate. I don't think it's that easy to replicate it, but eventually people will. But the mote of proprietary data and basically using that to do the lending business, which is a very old business and something that people are going to need a long time from now.

So I think that really is exciting. I think there's going to be interesting conversations around data, like GDPR. Who owns the data? Do people care about privacy? And all that stuff. I think we have yet to see how that plays out, because in many ways, our incentives are not fully aligned, right? So I'm curious about that, but I think that's a business model that could use a lot more competition and a lot more innovation and I expect to see exciting things come out of the lending space, even as kind of transactional things go away as kind of competitions, similar to WeChat Pay, Alipay, drives transaction fees to zero. So when you have zero transaction fees and no data, probably not. But I look forward to seeing how that kind of tension plays out.

[0:56:49.1] JM: Sure, yeah. I mean, I think you even look at the companies today, like Tala or Affirm and you already start to see really interesting ways that lending can be done with kind of the technology we have today. It's like they're not really doing anything mind-blowing in terms of technology. It's more like they've had the insight, like, "Oh! We could do this interesting thing and then we can make better loans." So I'm sure that's an area that you can get into.

[0:57:14.6] WB: A lot of financial services and kind of what is going to be disrupting the big banks and kind of the incumbents is really just execution, right? You don't need to be a rocket scientist to know that you should use modern open source technology and that you should write tests, and that you should kind of maintain and look out for the maintenance and technical data of your stack, right? It's easy to talk about it. It's actually really hard to have the discipline to do it

and to do it consistently over a period of years. So I expect execution to really be the differentiator.

[0:57:43.3] JM: Right. Totally. Okay, Ed. Well, thanks for coming on the show. I really have been enjoying talking to you.

[0:57:47.1] WB: Think you very much. It's been great.

[END OF INTERVIEW]

[0:57:51.6] 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.

[END]