EPISODE 1478

[INTRODUCTION]

[00:00:00] SPEAKER: We're happy to welcome back to the show, Cos Nicolaescu, who will be discussing technology choices made by Brex as it continues to scale at an incredible pace. Jeff and Cos discuss programming languages, databases, performance bottlenecks, and much more in this episode. As a reminder, Cos is the CTO of Brex, where they are building an integrated financial platform to empower both finance teams and employees to make their business run faster.

[INTERVIEW]

[00:00:31] JM: Cos, welcome back to the show.

[00:00:32] CN: Hi, Jeff. Thanks for having me again.

[00:00:34] JM: Last time, we talked about an overview of some of the systems that you built at Brex, and I wanted to start by going a little bit deeper into some of the choices of a technology for the stack. So, I wanted to start with database basically. When you think about the high volume of transactions that are stored across a credit card platform, there's lots and lots and lots of transactions, and in some cases, they have to be accessed with pretty quick latency. In some cases, not as sensitive. I want to know about your choice of storage for those transaction records, because I could see cases where you would want them to be stored in a transactional database and other cases where maybe you would want to just put them in giant parquet files. Tell me about the choice of storage for credit card transactions.

[00:01:35] CN: Yeah, so we chose SQL for that. Specifically, we're using RDS with MySQL. There's a couple of reasons that we did that. Number one, was that we wanted something that's reliable, proven, et cetera, and scales. And obviously, both MySQL and Postgres can scale quite a lot. The second reason why we went with the SQL technology was that we wanted to have local consistency, and this is where I think a lot of companies tend to bias on consistency in different extremes. For example, some companies are like, we don't care about consistency, because we have an SOA architecture, so SQL doesn't help. The other argument is like I want to enforce global consistency through SQL and it creates bottlenecks. We wanted to have the local consistency within different systems. So, we don't have a shared database. Every system, every service has its own database. Within that system, though, it is important to have consistency, and if you don't have a transactional store, then you end up having to do it yourself, where you enter a transaction, and then things related to that transaction in different operations, and you need to make sure there's a consistency there. Otherwise, you have to do that asynchronously.

So, we're not particular, gRPC call comes in, we will make sure that it gets saved consistently with that service. We then have a message bus that's through Kafka, where we publish particular events, different services can subscribe to those and asynchronously process those and then locally store them within their own SQL database. And then for anything that requires synchronous validation, there's subsequent gRPC calls that we make between services.

[00:03:17] JM: What do you mean by synchronous validation here? I mean, if a transaction enters the Brex system, isn't that a valid transaction?

[00:03:27] CN: For example, a risk check would be something that we would validate synchronously. So, a transaction comes in and we want to basically validate that it's not fragile activity. We take a lot of the metadata comes in with that transaction. And rather than having it through any synchronous event, where we can process it later, we need to do that synchronously. We know whether to accept or decline a transaction. There's very few of those calls.

Another example would be around time validity. So, if your account got suspended, for example, the suspension of accounts happens asynchronously in nature, because there are so many systems and so there are a certain amount of checks in different – or critical flows, where we want to make sure that it's not something that's been suspended. In general, we try to minimize those as much as possible, and primarily focus on building this asynchronously. For example, if we have to send you an SMS or push notification, if we have to take additional actions, if we have to update with models, all those happening synchronously via a message bus.

[00:04:23] JM: You mentioned a risk check there. I don't know that much about credit card risk infrastructure. To what extent is the risk or the fraud analysis handled by the underlying credit card network like Visa or MasterCard? And to what extent is it handled by the actual branded credit card?

[00:04:47] CN: So, in a transaction flow, you have three main parties. You have the acquiring processor, so someone like Stripe, for example. You have the network, Visa, MasterCard, American Express, and then you have the issuing bank, which in this case would be Brex. The first party, so the acquiring processor and the third party, which is the issuing bank generally take the risk in different scenarios. The network's Visa and MasterCard predominantly facilitate the networking or the routing for that. When a transaction comes in, MasterCard roll in this case, since Visa cards or MasterCard will wrap the transaction into Brex. It is Brex's responsibility to approve or decline that transaction. As a result, we're the ones who take the risk.

So, if we approve transactions that then turned out to be fraudulent, that's on Brex to do. What the networks do, in terms of involvement for fraud is both on the issuing side, and the acquiring side is looking at thresholds. So, if an acquiring processor has too much fraud, i.e. chargebacks, that will go against them. And over time, the networks will basically come down on them. Similarly, if an issuing bank basically has too much fraud, and similarly over time, there's a discussion where you can be part of the network, but they don't take – so the risk is pushed to Brex.

[00:06:03] JM: Okay, cool. Well, thanks for that slight digression. So, talking about the database and you also mentioned Kafka there, so what's the lifecycle for data stored in Kafka? Do you just – is there some default that you adhere to in terms of how long you leave data in Kafka? What's the protocol for how different systems utilize Kafka?

[00:06:33] CN: So, Kafka has two different modes of configuration, at least once delivery, or at most, once delivery. We basically have at least one. So, each system has to deal with duplicated data. Because otherwise, you reach a bunch of different scale issues, and so there's item potency built in so that we know if we've seen any better or not seen any event. The second choice that you have or we made is between reference data versus actual data. In the case of Brex, we chose to have reference data. So, we'll have all the IDs and restore any data that is pertinent to that particular event, to help with any replayability that we might need in the future. Which brings us to the last point, one of the challenges is, once you've consumed an event that you have to then replay it, because maybe you have corrupt data, maybe something happening after we build. and so we have the capability to replay events, which you can go and configure.

So, when that happens, Kafka has no idea initially what system has to replay them, so they replace the event, the systems that don't care about it can discard it ever, because it gives you an app item potency, and then the system that does need it in that case, would know how to replay it and it knows, you end up building custom margin for that. So that for example, if I have to know whether I need to resend a communication or not, it really depends. In some cases, you might have to, in some cases, you might not. And so, you end up building custom logic for that particular. We use migration terminology. So, whether it's a database migration or service migration, and so you'll end up building a custom migration where it's like, I know that I need to do this in this particular case and how to handle that particular replayability scenario. But it allows us to kind of bring back. It's kind of like an eventual consistency model where we can recreate the world as needed. And then those events actually don't remember exactly. There's also a limit to replayability. I want to say, it was 90 days, but don't quote me on that.

[00:08:25] JM: Have you ever had to run that world recreation operation?

[00:08:29] CN: We have. It goes back to whenever we encounter certain bugs. So, if you have a bug in your system, and they realize like, "Oh, I have a bunch of data that's now stored, that's inconsistent matter", you have the option to basically correct it within the database. But then it's really tricky. Because if you have subsequent events that get published, now you have to reach eventual consistency to the rest of the world. So, to the extent that you have to fix something just within your system, and no one else cares, we just do a database migration. We fix it up to the extent that it's a business logic change and impacts both yours and dependent services. We end up doing it through replayability of events, such that the rest of the world can know about those as well.

So, imagine that you want to change the state on an account, because of different events that have happened, you can change the state on your account very easily in a database. You can do a simple update query where you have whatever criteria you want to do for selection of the subset of data, and whatever the new status. The challenge is whenever you do that, there are other side effects from that particular change. We might send some communication. We might want to disable or change the state in other systems. So, either we would have a bunch of different database migrations that all have to do the same thing or we basically do it in a single place where we replay that, we change that logic, we replay the event and then the rest of the world ends up getting to be eventually consistent because it's as if you're processing things for the first time.

[00:09:56] JM: When you have a really, really large Kafka cluster are there any issues that you encounter that you do not encounter at a smaller scale. I haven't really done any shows where we went deeper on the scalability concerns of a large Kafka cluster.

[00:10:16] CN: I mean, you end up having scale issues just like anything else storage related, to be honest. So, you'll end up having event delays if you don't shard and configure Kafka properly. Those tend to be the main ones. We've also had some issues that we had to deal with in terms of tooling that we built on top of Kafka, such that you can like search for events, configure them, et cetera. And then it's less about scale, it's mostly about like how do you visualize such a large sculpt? If I just want to go through those, it's just like a large number of events.

Most of those have pretty good in terms of like, how do you set up your Kafka to scale and has to do with a topic configuration. We generally tend to be pretty thoughtful about what's a topic? And what level do we have it? Because if you end up having exclusion on that, then you end up having issues there. And then the other one that we have is mostly less about Kafka in particular, in terms of technology, but more about like how you process events. When again, you start having these dependencies where event A comes in, and that triggers event B and C and D. And again, there's no guaranteeing of ordering with Kafka, and so you might end up having events that come out of order, and you have to get the systems to handle those well.

The thing that happens in skills, is you just have more likelihood of those scenarios to pop in, and if you have any kind of bots who are not handling that properly, it's more likely to materialize. So, that one's less about a pure Kafka scale. But it is an engineering challenge for scaling when you're increasing probabilities of something bad happening. It's kind of the same thing with databases, like you can scale databases up and by yourself some amount of time where you don't have issues. And then at some point, you start having other types of issues and then you scale it up again, and you're fine until obviously, you can’t scale up anymore and it just increases the probability of something happening. If you take the example of database locks, you can have the wrong locking strategy and when you have a small number of transactional volume on that particular database, or that particular table, you're probably not going to hit any race conditions, where multiple branches want to acquire the lock. But the larger you are, the more likely that is to happen up until you get to a point where it happens on every single call. So, that's the same thing with Kafka where you're just increasing the probability for that to happen.

[00:12:30] JM: As the span of different platforms that you've developed has grown, at this point, you've got credit card system, expense tracking, spend management. There's a lot of different elements to the platform at this point. To what extent have you standardized the engineering process? Do you have strong dictums around tests or choice of continuous delivery system or choice of container deployment system? Do you have Brex CLI? What kind of stuff have you standardized on?

[00:13:09] CN: We do in general, the infrastructure is standardized. So, we do have a standardized CI pipeline. We do have a Brex CLI. We do have container strategy. We do have basically, like everybody uses kind of the same GitHub account with multiple repositories. It’s angelic, we standardized infrastructure pretty heavily. The reason for that is because you get economies of scale and it helps both with internal mobility, and in terms of s scale kill, like technical scale.

So, for example, if everything gets to choose its own database technology, then it's harder for engineers when they move from one point to another, they have to learn potentially a new technology. So, that makes it harder. Same with language choice, right? Every team choose its own language. Yes, you can do it if you have services and the way it talks to gRPC. But then if you move teams, obviously, you have to learn that. The second one is then about shared components, libraries. How do you do login business? How do you do access control, et cetera, or where? If every team ends up choosing their own infrastructure for that, then it becomes pretty challenging to kind of get that at scale. And the third one is in terms of expertise, where if I bring in a new technology, then is every team going to have a database expert on their team, for whatever technology they choose? And so on and so forth.

So, I found it useful to standardize on those. At the same time, I'm a big believer that you should have a small number of tools that solve the problem the best way. So, we don't have a single language at Brex. We have different languages that we use for different scenarios, because we know that there's a difference like we use Python on the day side, we use Kotlin predominantly now on the back end, we use TypeScript and React on the front end. And so, we try to use the best technology that we have on the infrastructure side. Similar with databases, like I said, RDS is predominantly used across the stack. There's a place where we have a graph database, because that has been useful in terms of like roles and permissions and access control. So, we do introduce new technologies. But when we do that, we basically say, okay, we're going to introduce the technology for this particular scenario, and then it becomes a supportive technology for these scenarios. It's not that anybody can potentially just go and use it just because it's there. If someone wants to write a business logic service in Python, they wouldn't be able to do. They shouldn't do that, and so the guidelines are there to help them with that.

The other aspect of consistency is around processes, and this is where I typically tend to think about as interfaces. So, we don't force every team to use Scrum Planning the same way. Some teams might use one week, some teams might use two weeks, some teams might have multiple sprints, and each will have a single sprint. And these might not do any expense at all, because they're just getting started and they're very small. So, it really depends, it's more about the interface. We have a plan cycle across the company, we have weekly updates, the team's published to the rest of the company across the different teams. And so, we try to find like, what are the things that really matter in terms of like, broader communication, versus enforcing that every team must do something a certain way? Because I don't find that particularly, and having that flexibility makes sense. So, try to find the balance rather than being too extreme on one way or the other.

[00:16:20] JM: You've done engineering at several companies before and this is your first CTO role. I'm curious if there's – we talked a little bit last time, you've worked at Stripe and talked a little bit about the differences between Brex and Stripe in the last episode. I am curious if there's any major distinctions that you've seen in terms of emergent characteristics of Brex engineering, that might not be obvious from the outside? If I look at it from the outside, it seems like you could manage Brex in many ways, the same that you would manage Stripe. Are there any peculiarities that are either emergent or planned that have stood out as kind of distinguishing the company in terms of how you operate it?

[00:17:08] CN: I would say, there's probably two main things if I compare it. So, I tried before and before that at Microsoft. In terms of both Microsoft and Stripe, they were very much in office. Both companies were distributed, as in they had multiple offices. But for the most part, even though you had some people who are remote, the culture was very much in the office, your team is there, most of the leadership team is in a single office. There's an HQ, et cetera.

For Brex, we were going on the trajectory that we had similar to Stripe where it's predominantly in distributed offices, and you can have some folks remote. And COVID happened and it forced everybody to be remote. Early on, we had really good conversations about what do we want that to be long term.

We realized that COVID isn't going to be something that's a few months, probably going to be like one or even more years long. And it turned out to be the case. So, what we realized is that, if that's true, then we have to make the company work in a remote world. There's no option. We have to invest in making remote great. Then once you do that, the advantages of being remote first, in my opinion, outweigh it, especially for something that's not super early stage. We've already given advice to someone getting the company off the ground. I would probably say you and your co-founder should start in person and plan to be remote.

But early on, I would say there's a known value of being all in one office and having that sort of collaboration. But once the company gets to be over a few people that actually think going remote, it's more advantageous. And those something that I've never done before, we're like the entire company, on the entire mission organization ends up being fully remote. We went through two iterations. The first iteration was everybody's remote and isolated at home, and you have to deal not just with figuring out communication, decision making, et cetera. But also, the psychology of it, where you go through waves. Initially, people are like, “Great, I don't have to commute to work.” Then they're like, “Oh, I feel really lonely and isolated.” And then they're like, “Oh, I'm not really set up to work from home, because I never planned for that, and so I just work at the kitchen counter and that starts hurting my back, and it's not great.” I'm basically going to the illusion of like, how do we actually make that experience great.

And then last year, we started thinking about, like, what does it look like long term once you are able to go back once vaccines came out? We started thinking about off sites and planning for teams to kind of get social activities together. How do we accommodate for folks who don't want to or cannot work from their actual home? A bunch of different scenarios for that. So, we went with – we worked passes and hubs in places where we have enough people if they want to have a co-working space that's together to kind of recreate that. But the culture is very much remote and distributed.

At the beginning of the pandemic, the entire leadership team at Brex was all in San Francisco, and now, it is pretty shattered. We have folks in LA, Miami, New York, Seattle, it's pretty broad and I think that also is a force in function to make sure the company can function well when people are not all in the same place. So, I think that's a big difference of like how do we make that remote first culture work really well, for the long term.

The second aspect, I would say, has been more about creating teams that are fully autonomous. So, one thing is, I tried to think about removing bottlenecks in the system, whether they're technical or organizational. Typically, your job as moderator ends up being removing those bottlenecks to make things faster as things scale. Whenever I get involved in decisions, I'm always like in mind the person that should be making this decision. I'm happy to make a decision now to get things unblocked. But like, if I think about the broader system, should that happen?

So, for example, if I think about a language choice, you introduce a language or naturally change the language. There was a conversation that we had, actually two years ago, we started on Elixir for the back end and when I first joined Brex, everybody asked me about Elixir and the choices there. And I covered that a little bit in the last episode. And then I said, we should reevaluate that later, based on how the language will mature, and the different types of bottlenecks they will hit with the language. And then once we realized that, like, the ecosystem of Elixir is too narrow, we started looking at different language options, and we ended up choosing Kotlin.

That decision was a decision that I ended up making and I'm comfortable with that, because as the CTO, it's reasonable for me to make a decision that impacts the entire engineering organization, it's so core to the organization. But there are many other decisions that come to me, mostly because it's the easier way and or where teams might not agree, or there's no clear odor. In those cases, I go look into like, okay, what prevented someone else from making that decision? And then go and look at addressing that. A big part has to do with like, how you structure the organization and how you set goals for those teams such they can operate more independently?

So, I'd say that's two, and somewhat tied to that is, how do we manage different functions? And this is something that I don't think many companies do. A lot of companies are like, well, are you engineering driven, or product lead or design lead or sales lead or whatever? And Brex is a very cross functional environment. Obviously, we have to deal with a lot of regulatory issues. We have to deal with operational issues. So, seeing that it's like your dollar sign function lead, ends up minimizing everybody else. So, we're very thoughtful and inexplicable, not being that sort of company. And as far as it comes to engineering, I spend a lot thinking about how do we help run teams across engineering, product design, data science, and so on. We really have gotten operations where initially like every company, you have like an engineering person and a designer, let's say an engineering director, designer, or product director, data science director, and they can work well. But everybody was responsible for their own kind of area and they weren't really running the team together.

And then about 18 months ago, helped flip that where I was like, yes, each of you have expertise in different areas, and obviously managed the function, and help grow people and support people in your function. However, like all of you, whether it's like three, four, because it depends some things might have design, **[inaudible 00:23:12].** You end up - you all are responsible together. So, for example, one concrete change was how we think about headcount. I used to think about ways to do headcount across like, engineering gets this much, referring director here, some headcount. If you're a product director, here’s your account. If you’re a design director, here’s your account. Go nuts. And we flip that where we said, no, you all as this group, get this headcount and you all decide how you allocate it within the different functions. And obviously, if you disagree, feel free to escalate and happy to be average for that.

And when that force and that fix is making sure that there's the right support and ratios between functions. So, there were teams were basically, maybe we had too many front-end people, and not enough designers or vice versa, or you might have an engineering manager, but not a product manager. And they weren't aligned on hiring priorities, because everybody was optimizing more for their own function. The moment we flip that, we got everybody to feel accountable for everything, all the way from hiring, to technical decisions, the product metrics, design, quality, and so on. So, I think that's something that's really unique. I haven't heard of a lot of companies that not only structure things that way, I think that's more common. But the way we run and we put kind of shared accountability across the leaders for those, and it's allowed us to scale quite a lot through this autonomy across different functions without having to go through different kind of organizational models that have their downsides.

[00:24:35] JM: You mentioned that you have a data science director, and so maybe you don't have as much purview into this. But we've talked about some standardization across the company. How much standardization do you have around data science and ETL and large-scale aggregation operations?

[00:24:54] CN: Yeah, so the data organization actually as part of my organization, and then data itself is organized in three teams. There are data science teams, data platform teams, and then data science is broken up into machine learning and analytics. That's something that I think has worked really well, by having those teams together. You don't have some of the arguments you have between the parts of data. So, we've standard these Airflow, and then data gets pushed into snowflake, and then we have Looker on top of that for metrics and that's been useful to standardize. We've always had that standardization. It's not something that I brought from beginning, but we've changed some of the technology that we use. I would say, by having a closer to engineering, and then embedding the data science folks in different parts of the organization, it's helped us with kind of ETL process where a team owns the underlying model, and then it makes them change, and then it ends up breaking ETL jobs. And even if you don't break them, people don't understand what it is. It’s like, okay, what is this new field on this model?

So, it's helped us kind of have a more visa vie, because the scientists that is embedded on a particular team has much more knowledge of the domain stare, and then can kind of bring that knowledge more centrally within the data org. And you have the kind of career development from having a centralized org and functioning in terms of ability that you get from being you get, to try to kind of find a balance between both worlds.

[00:26:18] JM: So, is Snowflake been standardized as the place to throw all the data and do those aggregations?

[00:26:23] CN: Yep. So basically, data gets pulled from SQL through Airflow. We push it into Snowflake, and then Looker sits on top of Snowflake. So, you can run custom queries on Snowflake, which is more than within engineering, and then you have basically all the reporting in the company that's done on Looker on top of that.

[00:26:45] JM: I'm always curious about the Snowflake bills. It's kind of, in some ways, I think, kind of like the new Oracle, not in a bad way, but it can be quite costly. Do you get involved because that's like a big enterprise deal, I'm sure you have. Do you get involved in the like negotiations with Snowflake to figure out the pricing?

[00:27:08] CN: So yes, to some extent, as any technology procurement that we have, whether it's AWS, Snowflake, Looker, Okta, Salesforce, et cetera, et cetera. I ultimately have to approve those. We have different teams that own it. So, for example, for Snowflake, we've had books on the data platform side involved. For AWS folks, infrastructure side involved in the negotiations. We also have someone on our finance team that works on procurement, who helps with contract negotiations. I will use this third-party vendor who also helps us negotiate contracts and larger scale. So, definitely for like high value contracts, we're pretty involved in negotiations.

The thing that I generally look for is, there's obviously like scale. Basically, a lot of these contracts, you end up having multiyear contracts and your pricing is determined based on the scale that you have. So, obviously, I get involved into thinking about like scale for Brex more broadly. And then from there, there's more of the unit economics. And then generally what I look for when it comes to some of the spend is, how does that spend grow relative to our revenue? Basically, if I think about pressing that we have, how that grows for the business, and how does our infrastructure cost grow across the board for all these things, as long as it's flat, or ideally goes down, it doesn't go up, that I think you're in a pretty good place and it's probably not worth optimizing for that, because ultimately, you end up spending a lot of time and it slows down your business. But to the extent that you're basically like your costs are going up higher than your business growth, then you have a problem.

So, either you have a unit of economics problem, or you're just paying too much, and everything goes through the contract. More often than not, though, you have an inefficiency problem, where are you just using resources in ways that just don't scale for your business. And then those are the harder conversations where you have to reevaluate how you think about that. But in general, like when you are super early stage, I typically tell companies to focus on the architecture and less about on the costs. So, as long as your architecture scales, you can work out the unit economics later. You also have more leverage in negotiation, so later on, when you are super early stage, and you're a small fish, that you're just not going to have a lot of leverage negotiating with someone like AWS, or GCP, or Snowflake.

But as you grow, and as they see the growth that you've had, and it's demonstrated growth, not just hypothetical growth, then they're much more interested in negotiating to make sure they get your business and they're sticking with it. And then two, in terms of amounts like, yes, your cost might be growing high early on, but what if you're talking about thousands of dollars, tens of thousands of dollars, or even hundreds of thousands of dollars a year, when you're a venture backed company and you've raised tens of millions, it's not the place where you get the most leverage. I've never seen a company that fails because of that. It has to be on a value prop market fit, as it's a scalable architecture. So, it's one that you can –if you're talking to my esteemed economics, then those costs would be in line to your business growth.

[00:30:10] JM: Gotcha. Is vendor management a big part of your job? Like negotiating those kinds of things, or selecting vendors? I guess the vendor selection is probably more of a lower level thing. But yeah, maybe you could talk through vendor selection, vendor management. And if there's any good anecdotes you have recently of vendor selection or vendor management, I'd love to know.

[00:30:32] CN: My involvement generally is we have across the company, a policy. We thought about how do we actually scale that such that we empower people to make decisions rather than centralized too much. So, we're like documenting our own products and that sort of stuff. In general, like any of these vendors that you basically have commitments up to certain amount of years and dollar amounts, there's different approvals that happen in different levels. Some manager didn’t have, some you need a director, some you might need to approve, and some might have to go to the founders. But in general, like at this point, it mostly stops at each of us for our own functions.

For the large ones, my general involvement is less in the selection, because I think we've hired really, really smart people and they have a lot of expertise in those particular areas. So, they should be the ones deciding. Obviously, we have quite a few processes, and we look at selection, and so on and so forth. In general, we also have our legal team, and again, the finance team who's involved in the logistics of it, so we don't have engineering having to spend a lot of time on that. I don't think engineers going back and forth negotiating on the price specifically, is the best use of time. And then similarly, going through contract negotiations and reading contracts and redlining contracts is not something I would necessarily want to have engineers spend time on. And so, that process is handled as part of a workflow by some of these other functions.

Then the final version, again, if I have to approve it, because it needs a certain threshold comes to me, so I ended up reviewing the contract and mostly skimming through it, because again, at that point, it's kind of gone through enough reviews, and then signing it for the company. In terms of anecdotes, I think the most interesting one was on Docker recently, because Docker went through pricing changes, and it changed their pricing model. But it didn't seem like it was really well rolled out. Because there was still confusion between different people that you talked to the company, and just like there was a lot more back and forth, for something that was very obvious, like companies use Docker.

So, one of the choices of technology, and they had a pretty easy way to kind of roll that out. And I remember spending some time with the folks on that content being like, “Hey, the fact that we're spending so much time on something that seems very basic, not to trivialize it, but like, it's so hard to do everything, seems like there's some problem there.” Other than that, in general, the process is pretty smooth. The thing that we're always looking for is if there's bottlenecks. One of the things for the procurement team on site is basically like, what's the end to end process? So, if I come up with a new vendor, I say, “I want to use”, let's use Docker as the example, how fast can I get it done? And it's hard to monitor for that, because there's obviously in a lot of these vendor negotiations, there's back and forth.

You set time or the other team, you go back and forth, you have lawyers involved, et cetera, et cetera. So, it's not like purely technology where you can have full control and optimize everything. But we generally tend to focus on what are the things on our end. We measure how much time was on our side, and how many times it had to bounce around or back and forth, to try to see if there's a way to optimize it. And then the other one that kind of comes with scaling companies is we created initial thresholds and policy back in probably, I want to say it was 2019, or maybe 2020. So, we weren't like super early on, or we just didn't have any policy in place. But we were probably a few hundred people, and those thresholds never got updated. So, there were times where basically for something that's like, relatively small amounts that there are scales, like, I would have to approve it. And again, I don't mind doing it. It's just like, creates another bottleneck.

So, in the interest of removing bottlenecks, we ended up changing those. And I was really happy as a company, because you hear all these horror stories from companies just like procurement, and such a nightmare, or this bureaucracy and red tape. We literally like had a half an hour conversation, then the next day, there was a proposal with here's how we think about updating limits, et cetera. And then we rolled it out the week after, those really, really streamlined and it kind of helped us scale as we have more management layers and more levels and different thresholds for analysis in terms of materiality. So, I was quite happy about that.

[00:34:37] JM: Well, one thing I don't think we talked about last time was banking integrations. I'd like to know, you obviously don't have to have the constraints of a bank, because you don't have to hold large sums of money like a bank. But you have a lot of the features of a bank like actually sending money and managing credit cards, and there are use cases where you have to closely integrate with a bank. Can you talk through, I guess, for some of the prototypical banking integration use cases and some of your strategy around how to integrate with banks successfully?

[00:35:17] CN: Yeah, I think just like on the credit card side, there are two paths of doing it. The same thing is true on the banking side. There's the path in which you partner with someone, and they handle everything for you, and that could be someone like modern treasury, that can be someone like a single bank that basically you are the UI layer on top of it, which is what Mercury does, and a few other new banks. And basically like, the bank is the one that processes the money movement has instructions, has to control it. It's not like you have a very intuitive presentation layer.

The second option is when you go deeper. I guess there's a third option and you become a bank, and that's what Square did by getting a banking license. The second option is when you basically have a banking core, you're the one that manages the flow of funds, initiation, and you take on more of the risk. And then you work with one or different banks for actually submitting files and getting files processed, or wire files process. That's the world that we're in.

So basically, Brex is pretty full stack. And then we integrate with the banks on two levels, one from a licensing perspective. So, they basically sponsor Brex, and then we have viability. But as far as a kind of broader environment, the banks are liable for Brex's behavior. And then the second piece of integration is for actually integrating with like Fedwire and NACHA, to actually move money, and they actually hold the funds. So, it's not like Brex is in the bank, we have a bank account, the funds sit with an account within that bank, and we manage basically those balances.

The advantage that we have with the choice that we made, obviously, like the banking license is something that's like, incredibly complicated from a regulatory perspective and it's actually very rare to manage those. And once you have it, you have a ton of overhead. So, you really, really have to be very thoughtful if that's actually worthwhile for your business.

The first approach is generally the most common and simplest. Again, most of the **[inaudible 00:37:11]** end up doing that, because it gives you the most fastest speed to market where you build some UI there, you're going to go with some files, and then you're good to go. But you do bring some value for the customers in terms of a simple user experience, which most consumers would care about. The business side, it's a little bit different, where one of the things that people love most about Brex, it's just how well the card, the cash account, the Empower platform, everything works well together, which is really hard to have when you have different systems and completely different decks.

So, if you work with like a card processor, let’s Stripe issuing or Marketo on one side, and then you work with a bank on another side, and so on and so forth, you can't really tie those together. Because we've built the infrastructure on both sides and more holistically, you can earn rewards both through your card and through cash. You can have a single card there works across. You can make sure that management works really well with both, that you can pay bills and things work well together, again, both ACHN card, and so on and so forth. And so, what we've done is we actually have integrations with multiple banks for different scenarios. We recently started working with Column, which is a bank that's built by one of the plaid co-founders and that's been really positive for us. In terms of the developer experience we have internally and the operational load, we also work with JP Morgan, where we have wires, so a lot of international buyers that we send code through JP Morgan. And then we're also doing integrations with different partners in terms of money movement around the world, to kind of improve that experience and to start having more local payments.

So, if you're an employee, in, let's say, the UK, you don't necessarily want to get reimbursed through the wire, because it's not a great experience, it's obviously much more costly. You want to get reimbursed through your local bank account or your local payment method. So, we were using operator with your partner to basically facilitate the money movement there. So, the way I would think about it is similar to the card side, you end up having banking core, and so you kind of orchestrate the money movement, you own the presentation layer, so the user experiences are polished. And then you're able to get the leverage across the platform and making things work really well between the different parts of the product. And then you're able to plug in different partners and work more broadly, to have the best user experience for customers in different situations.

[00:39:29] JM: So, I'd like to close off with maybe you could give a recent anecdote that kind of illustrates your role as CTO, maybe a firefighting situation or an architectural decision. Any anecdote that stands out in terms of describing and articulating what some of the more important decisions you have to make are.

[00:39:56] CN: I think, one of the places where it’s much – like I said, I really value creating autonomous independent teams. At the same time, I think there's always a balance between short term and long term. And if you optimize for the long term all the time, then most of times you end up not having a fair amount of time. So, that doesn't always necessarily mean it's the right answer. And if you optimize always for the short term, obviously, you're just going to have a ton of hacks and technical debt, and then eventually, you crumble. So, every company tries to find the balance and kind of iterate through it.

One of the interesting things with creating these autonomous organizations that each focus on their own things is exactly to your earlier point, less about infrastructure, which is like which parts are standards versus not, but more about reusability, and platformization. So, a recent example, in the Empower platform, we have rules, and you can build policies. And these policies can be quite complex. You can say, I want transactional, like the secret reimbursement over certain amount to be reviewed by this person's manager. And over another amount, you have another tier of management that want for these expenses to have a particular level, because I think, we can pull that from your HR system, and so on and so forth. And you can create, like, we've built an engine that's pretty flexible.

There was a debate between whether we built that from scratch, or we similarly had a risk engine that we basically use for risk rules for our fraud system. And that was similar a kind of a generic engine for rules, and you can get them evaluate them, you can collect data through them. Obviously, there's always this tension, because on one hand, you never had this era where like, everything fits perfectly. So, you look at it and this is the same thing when you have external technologies, you look at something, it's like, “Oh, buy versus build.” I looked at buying the solutions, but none of them were exactly what I want. So, I'm going to go build my own. More often than not, if it's not core to your business, it ends up biting me in the long run, because you're not going to invest efficiently enough when it's not, again, core to your business.

So, for buy versus build, we've always aired towards like, we should build everything that's core to the business and we should buy that’s not core to our business. And so last year, we moved, we used to have our own user authentication systems, all logins, everything were in Brex. Now, we move to Okta, who offers that as a service and they're going to do a much better job in terms of delivery for two FA in terms of more complex rules, in terms of detecting irregular activity, then we will, because it's not core to our business, and we can make those investments and we can help people build it. But again, you never end up having a positive ROI relative to the opportunity costs of investing in something that's core to your business.

Internally, you end up having somewhat debates where the team that worked on Empower was looking at the policy engine that was built for fraud. And they're like, "Yeah, but it doesn't do the things that we want. So, we're going to build our own because we don't really need all those things." And that was a place where I ended up chiming in and literally looking through a code because everybody talks at a very high level of like pros and cons. So, actually going through the code to understand like some of the concerns, and then two, making sure that if we were to reuse it, that we have the right support to build whatever the gaps are. This is the advantage of internal versus external, is internally you to have more control over how you prioritize things.

But it was very interesting, because like, on one hand Empower team was like, "Well, if we want to hit all these deadlines, we only need a subset of this and it's easier for us to build rather than take the dependency and have those risks." And then the risk team was like, "Hey, we build this as a piece of platform and infrastructure that can be used more broadly, it just seems wrong for us to just do multiple of these across the company." No one was wrong. There are few pros and cons. But that's an example where like, I had to go in and not just consider because like you can make it on the theoretical basis, but actually go and literally look through all the code to understand kind of the state of the world and understand like what decision would you make as an engineer.

It's very hard, I think to do, because I don't write code on a daily basis. In fact, I rarely write code outside of hackathons. And so over time, you get more rusty on that, so you're not going to make necessarily the best technical decisions. But having the ability and having been short enough in terms of technology to be able to kind of dig into unnecessary and make these decisions, I think ends up being something that I really value in the CTO role. And when I look at other CTOs of other companies. The best ones tend to be the ones that have found a way to remain involved in technical decisions, versus purely the organizational aspects of the role and then you delegate everything technically.

The other thing to that is basically like how do you make sure that that trickles through your management layer, because there used to be a time back in the day when management was all about the organization and people side, but realistically, the best engineering managers are the ones who are still pretty technical, and they find the right balance such that they don't have to make the technical decisions for everything. They don't have to write code for everything. But they're able to support people and they're able to know what to dig into and they're able to know when they should make a decision because there's too much debate going on. You need a tiebreaker at some point.

So, figuring that out organizationally, not just in terms of the hiring process, in terms of expectation setting for technical positions and the balance between life and engineering, but tech lead, a manager, and when different people should jump in and make different decisions and how do they get involved is something that I spent some time towards the end of last year as well.

[00:45:09] JM: Cool. Well, thanks for your detailed answer. Well, Cos, it's been a pleasure having you back on the show. I always enjoy talking to you.

[00:45:15] CN: As always. The same here and look forward to chit chatting more in the future.

[00:45:20] JM: Likewise.

[END]