**EPISODE 1515**

[INTRODUCTION]

**[00:00:00] ANNOUNCER:** This episode is hosted by Alex DeBrie. Alex is the author of *The DynamoDB Book,* the comprehensive guide to data modeling with DynamoDB. As well as *The Dynamo DB Guide*, a free guided introduction to DynamoDB. He runs a consulting company where he assists clients with DynamoDB data modeling, serverless architectures, and general AWS usage. You can find more of his work at alexdebrie.com.

PostgreSQL is a free and open source relational database management system. Postgres-based databases are widespread and are used by a variety of organizations, from Reddit, to the International Space Station. And Postgres databases are a common offering from cloud providers, such as AWS, Alibaba Cloud, and Heroku.

Neon is a serverless open source alternative to AWS Aurora Postgres. It's separate storage, and computes, and substitutes the PostgreSQL storage layer by redistributing data across a cluster of nodes. Today, we spoke with Nikita Shamgunov of Neon. We discuss how Neon scales Postgres, how it saves cost, and the engineering that makes it possible.

[INTERVIEW]

**[00:01:21] AD:** Nikita Shambunov, welcome to Software Engineering Daily.

**[00:01:23] NS:** Happy to be here.

**[00:01:25] AD:** Awesome. Well, I'm super excited to have you on this one. I've been in the serverless space for a couple of years now. And I think the biggest thing I've hear heard people ask about for a long time is like, “I want a serverless SQL relational database. I want some of that.” And that's what you're kind of going on.

So, just background for people that don't know you, you’re a cofounder of Neon. That's we're going to be talking a lot about today, which is this serverless Postgres database. Before that,

you're a cofounder of MemSQL, now known as SingleStore, which is a really interesting in-memory, sort of HTAP type database as well. You also worked at Coastal Ventures for the last couple years. Still there. Incubating Neon from there. So, why don't you tell us what is neon?

**[00:02:01] NS:** Happy to. Yeah. So, the short description of Neon is serverless Postgres. Obviously, the world knows what Postgres is. And it's one of the most popular open source databases. And that's the one that is growing its share still, right?

If you go to dbenginerankings.com and look at the top five databases, and I'm going to call those databases commodity databases, basically the ones that are kind of the default choice for building apps, you will see that out of the top five, Postgres, today, is number three, I believe, after Oracle, MySQL, and SQL Server. And then after Postgres, there's Mongo. And after Mongo, there's Redis. And within those top five, Postgres is the one that's growing share if you believe dbengineranking.com.

But kind of intuitively, it kind of feels that way, right? If you go and read Hacker News, you will see more blog posts about Postgres than about those other databases. And certainly, you also see it's in the ecosystem. So, the ecosystem is building more and more extensions, more and more ORMs, more and more – I don't know. GraphQL frontends for Postgres. So, clearly the world is excited.

So, what is Neon? Neon is taking Postgres and making it serverless. And when you think about it kind of as an engineer is, what would I do if I wanted to make a post go serverless? Well, people will probably start thinking. It's like, "Okay. Well, maybe I put it in containers, put it in Kubernetes." But then what? It's a stateful system. And because it's a stateful system, as you're changing the size of the container that Postgres runs in, or you're moving that container from one place to another, you will need to move the data.

And so – and data is not that easy to move. I mean, it takes time, it has certain amount of affinity and gravity to the place that runs there. You can also say, "Well, I will completely redesign everything from scratch, and I will build some sort of routing system, and then requests will choose where they go. And that's how I'll, kind of in the multitenant way, allocate resources

per request.” Well, that's go anyway from Postgres. That is a brand-new system. And there are systems like this out there, DynamoDB, or Cassandra, and whatnot.

And those, if you squint at them, you will see that they're mostly key value. They're not allowing you to run full-fledged SQL. And moreover, they'll not allow you to run full-fledged Postgres with like all its extensions, and sophisticated compute, and store procedures.

And the observation is that a key value request is bounded in the amount of resources it consumes, versus a SQL request or a transaction. And especially multi-statement transaction is not bounded with the amount of resources it consumes.

So, the enabling technology that allows you to make Postgres serverless is separation of storage and compute. And we'll certainly talk more about it. Once you separate storage and compute, you have an extra degree of freedom where you can put that compute. This node or that node. And by moving, you can move it – compute a lot faster. And that what allows you to build a serverless offering. So, in short, Neon is serverless Postgres. It achieves that by separation of storage and compute, and gives you additional capabilities, additional features for you to build and run your app.

**[00:05:31] AD:** Awesome. I love it. So, two really interesting things on there I want to follow up on. First, you're talking about the growth of Postgres in particular in this area. What's accounting for that growth? Why is it gaining share compared to MySQL and other ones?

**[00:05:44] NS:** Yeah, I really thought about it deeply. I've been observing that for years. I think the moment that Postgres really became visible and known to every developer was Heroku. And I think it goes back to – I don't know when. 2009 maybe. When Heroku introduced Heroku Postgres and made that the default database. That became like a distribution platform for Postgres.

But if you also step back and think what are the contributing factors for growth of Postgres compared to MySQL? There are architecture and license choices. Postgres has a Postgres license, which is the most permissible license on the planet. And Postgres is not owned by

anyone. In that way, it's very similar to Linux. By design of the community, Postgres community, it is impossible to take ownership of Postgres. And that's a good thing.

As opposed to MySQL, that, first of all, it has a GPL license, which is less permissible, right? It is contagious. So, you think twice before bringing MySQL into your organization. You have to run it by your lawyers. It's not like GPL licenses is something that's particularly new. But it still adds a certain amount of friction, where versus Postgres, in a way, is in no regret.

The second is very database specific. Postgres chose to be squeaky clean with regard to standards. And so, the surface area is richer. And the way Postgres implemented those features are “by the book”. And that is important, because it doesn't surprise you.

It places MySQL, and I happen to know MySQL very well, because Single Store implemented MySQL wire protocol in a type system. When you look at the type system, at times, you're like, “Whoa, that's a weird decision here. That's a weird decision there.” MySQL didn't support schemas for a long time. MySQL didn't have full SQL surface area with Window functions for a long time. I think it has it now. And all of those things are just like a no event for Postgres. Postgres has it all.

I think that there was another big deal for Postgres of, again, being kind of purists towards open source and standards. That over a long run, and database projects tend to spend decades, adds up to develop a trust and community.

I think there's a third thing about Postgres, which is extensions. And extensions allow you to build a massive ecosystem. And so, when people think about Postgres, they don't think about just the database. They think about just the database and the full ecosystem of extensions that you can bring with the database. The most popular are our PostGIS timescale. There is a handful of others that are making the top five. Very easy to find out on the internet. But extensions are a very, very big deal.

**[00:08:42] AD:** Yeah. I totally agree. Those are great points. And I think that distribution point you made – I think of when I started programming, it was Django and Heroku. And because of that, both of those, very Postgres heavy. That was the first database I used. And honestly, now,

for the last couple of years, been doing a lot of serverless stuff in terms of AWS Lambda, and using a ton of DynamoDB. And it's amazing how much sort of your compute platform sort of determines your database choice in some of those ways. At least just pushes you in sort of default ways.

I like that point you made about the extensions too, and PostGIS, and timescale. Those are ones I think about. I know AWS restricts things like which Postgres extensions you can use with RDS and things like that. Does Neon allow sort of bring your own extensions? Or what's that look like especially as a manage provider?

**[00:09:24] NS:** As of this point in time, we bless a handful of extensions. And PostGIS is certainly one of those. In the future, we would allow you to bring your own extension. And people are already asking for it. Because some extensions, we just didn't compile. Actually, it's not that hard to support. It's easy to compile those extensions in. But many people actually build their custom extensions. And those, we actually want to support. And what we're thinking of allowing people to do is to give them the ability to compile their extensions into a Postgres instance themselves and then push it as either a Docker container or maybe a micro VM, probably Docker container, into Neon.

Underneath will translate the Docker container into a micro VM anyway, because we would – for custom extensions, we would need some sort of VM boundaries around it. But that's, I think, what the vehicle is going to be in the future. We're committed to supporting custom extensions.

**[00:10:27] AD:** And just going back then to that serverless point, Neon being serverless Postgres. That term, it's sort of overload, and everyone's trying to grab a piece. But when you think of serverless Postgres, what aspects of serverless do you think Neon has that makes it serverless?

**[00:10:40] NS:** Well, the most important – There are two things that come to mind. The first one is you don't think about sizing. So, you say, “I need Postgres.” And in two seconds, you get a connection string. And that's all your interface with the service. Then within that connection string, behind that connection string is a Postgres. Is Postgres. And is Postgres compute.

What's the size of that compute? In the moment, you can query it, right? Via Postgres system functions and system tables. So, you can find out the size of that compute. But we will change that, the size of that compute, behind the covers, based on the intensity of your workload. We will also scale it down to zero if you don't use that at all.

And today, cold start is about two seconds. And I think, over time, we'll reduce the cold start. I think the lowest bound that we can push it to is 100 milliseconds. But yeah, we'll keep making investments in this area. So, not thinking about sizing is one. That what makes it serverless.

The second thing that makes it serverless is consumption-based pricing, right. So, we can have subscription-based pricing, or we can have consumption-based pricing. And subscription-based pricing is what you typically see in many, many database products. When you say, "Well, you have a database of size X. And this is going to cost you X dollars a month." You see it all over the place.

What I think is people want is a consumption-based pricing, where if you're using more, you're spending more. If you're using less, you're spending less. And then you're efficient. Our interests are aligned. Those idiosyncrasies, it really come in once you start rolling out your sales team. And if you are a subscription, then the sales team is very incentive to sell you a bigger and bigger subscription, which oftentimes ends up being shelfware, which over time leads to churn, because like your users are not idiots. And they do the math in their end.

Consumption-based pricing I think is universally better for infrastructure. And it allows users to – it just simplifies their lives, right? And then you can potentially get discounts for bigger commits and whatnot. But that's really going into the sales motion. I think consumption-based pricing is strictly better for the user. But you can only deliver on consumption-based pricing if you are serverless and if your own infrastructure allows for a consumption-based usage.

**[00:13:15] AD:** I love that point about the alignment of incentives with consumption base. And you can get the short-term win with that subscription-based pricing. But it's sort of like you can shear a sheep many times. You can only slaughter it once sort of thing.

In terms of consumption-based pricing, what does that look like? Is it like playing a scale where it's based on number of rows read? Or is it based on how many compute instances are running? And that sort of scales up and down based on what you're doing? Or what's that look like for Neon?

**[00:13:39] NS:** Yeah, we haven't rolled out our pricing. And there are intense debates that are happening internally about what it's going to look like. It will be consumption-based. That is absolutely the case. But what are the exact details.

Here are the things that we do know, right? We will charge for storage and compute separately in a way Neon is bottomless, which means if you go and create your database, and then you get a connection string, and you keep pushing data through that connection string into Neon, you will never run out of space.

And for us internally, for an empty database, we'll probably spend one page. And then for – which is eight kilobytes. And for a very large database, we'll spend terabytes of space. And the cold data of those terabytes of space, the ones that you might not be using, then we'll flush that down to S3. And we'll absolutely pass those savings into the user, right? So, storage is a lot cheaper for Neon than for everybody else.

Again, it doesn't make Neon cheap. Like, it's not going to be a cheap product. It's going to be efficient in the way it consumes cloud resources. And some of those savings will be passed to the passive user. And then we'll charge separately for compute and bandwidth. Unfortunately, I would love to not charge for bandwidth. But you kind of have to. And if you have an endpoint that points into the internet, that's part of your cogs on the cloud, which you run with a cloud provider.

And then with regard to compute, I'm not sure I love rows read and rows written. It's certainly one of the options on the table. What I like so far is that, really, on a per second basis, there is a certain size of compute we allocate into the user. And so, we can just add up all those seconds, add certain amount of margin, obviously, for providing value and running the service, and then give it back to the user. And that's very similar to what it costs us. And I think it's relatively straightforward to understand.

So far, my head is in doing something like this, where you know that you will never spend more than the absolute largest instance multiplied per second, multiplied by the number of seconds you're using it. And of course, it scales down to zero. So, it caps the amount of spend that you have. But then you gain efficiencies by not using the largest instance all the time. So, that is where my head is at on how to charge for compute. But we haven't ruled out pricing yet. And so, there's still some intense debates going on.

**[00:16:15] AD:** Yeah, it's interesting with the move to consumption-based pricing. A little bit of that unknown and like having to reeducate people on it, where previously, they just buy a giant instance, having a low utilization most of the time. But at least they knew what their bill is going to be. And now it's like, "Hey, this is going to be cheaper for you. You just have to trust a little bit and understand what your metrics are."

**[00:16:32] NS:** Yeah. I think that we know, for sure, that branching will be free. As many branches as you want, because branches are a part of our architecture. They're implemented at the storage tier. And so, creating a branch is free for us. It should be free to the user, for the user as well.

**[00:16:47] AD:** Awesome. Awesome. I'm super excited to get into the tech around branching. One last question on product base. You mentioned like pricing still rolling out. I know, y'all, it's September 2022 right now. You're in private beta. Do you have any sort of timelines on when you'd like to get it public? And GA? Or what are you thinking there?

**[00:17:03] NS:** So, pricing is going to roll out early Q1 next year. I think we're pushing 2000 users on the platform right now. So, very excited to have that and just starting, because we opened the gate June 15th. So, we keep onboarding users every day. Probably around 50 users every day. And more and more users sign up. So, we're excited about it.

What's gating that really are two features. First of all, we need to have pricing and billing, right? We need to roll up what our pricing actually is. It's one of the most common questions when we talk to our community. The second thing is we teased everybody with branching on the front page of Neon, on neon.tech. And branching is there. It's there. You can consume it via an API.

But we don't have beautiful UI that supports it, where it's like literally being built right now and rolled out. And we're running user interviews.

So, we want to make sure that branching experience is very understandable for the user. I kind of want to emphasize this point. When you build something that people kind of relate to, which is like Postgres. So, I make an API call, I push a button, I get Postgres. Nobody's confused there, right? It's very easy to understand what that is.

Branching changes the way people think and interact with the database, at least somewhat, right? You have your main branch, and then you can create another branch and then create a full fork of the database. And now, because storage and compute are separate, potentially, you can create multiple compute and clients and assign them to branches. So, that introduces complexity. It's not particularly rocket science. But we still want to run it through the user interviews and make sure that people like really, really get it and love it.

So, the infrastructure for branching is there. And it works. And there are people using it every day. But that UX, and DevX, and API is being iterated on. So, that's the other thing that is gating us to just open it for everyone.

We think Q1 is where we'll open the flood floodgates. And we're also working with a few partners that can consume. I can't really disclose it just yet. But we're going to be rolling it out very, very, very soon, where you'll be able to push a Neon button on somebody else's platform, and that will instantly create Postgres for you. So, for those platforms, that will be knowing by gate.

**[00:19:28] AD:** That makes me go back to Heroku as well, and how you could spin up different providers, and all those different things, and just how useful that was to spin that up. I also love that point on branching, especially like when you see technology where there's like a step change in the speed of how something works, like getting a full copy of your database, just how that completely changes the use cases around it, right? Like when we went from deploying once a quarter, to once a day, or hour, or minute. It just changes all sorts of things that you almost came and predict and need to think about reworking a lot of different things.

**[00:20:00] NS:** Absolutely, we'd also think of that if the moment we have branching, that would allow people to also publish read only versions of their database and potentially accessing them kind of like GitHub. You know, there are public repos on GitHub. And you can have public Postgres databases. And you can go and either query them read only, and you bring your own compute. So, it doesn't cost anything to the person who published the database. It costs something for the person who is querying the database, which is how it should be. And then it allows you to push a button and fork that.

So now, you can publish a WordPress database. You can publish – or you can create a bunch of templates that are only visible to your organization and they become kind of starting points for people to build apps or starting points to run tests and Cis. So, I think lots of things can be done very – Lots of very cool things can be done with that separation of storage and compute.

**[00:20:54] AD:** Yeah, awesome. I love it. Going into that, I want to go deeper on the technical stuff, because I'm sure you have a lot of great stuff here. I mean, first of all, you're a multiple time tech cofounder of like hard infrastructure stuff. I love this pin tweet that you have on Twitter. So, talking about the three-step formula for infrastructure startup success. And the first one is, “Hey, find a 10X architectural advantage somewhere in cost and speed in a large category.” So, some sort of technological change and unlock something that wasn't available before. And then just build a really good team to exploit it and really focus on the developer experience. First of all, for Neon, I think he mentioned it earlier. But what's the 10X architectural advantage that neon has that you're exploiting here?

**[00:21:35] NS:** Yeah, great question. The answer here is – And I'll go kind of step by step, right? The architectural advantage of separation of storage and compute, and integration of storage with S3, right? S3 is like a big component. And if you read our blog posts, you will get more details about how this works exactly. How we maintain low latency for the storage. But still offload cold data to S3. That is our 10X advantage number one.

10X advantage number two is like what it allows us to do. And this is serverless. And we implement serverless today by orchestrating containers. So, that what allows us to put Postgres in a container. We're actually exploring micro VMs lately. Firecracker and Cloud Hypervisor are

extraordinary technologies. They allow us to do live migrations. Well, Cloud Hypervisor. Not firecracker. But Firecracker, you can freeze the world, and then move it somewhere else.

And as you do it, it's possible to not even break the TCP connection. Serverless, and that serverless orchestration is the other part of the 10X architectural advantage, because it makes things so much more efficient in terms of not over-provisioning things. So, that's our 10X architectural advantage. That architectural advantage is multiplied by the fact that everything is open sourced.

And open source is the contribution to trust, right? People build trust, because the technology is open source. It's a contribution to potentially partner with people who want to bring Neon somewhere else, not just the three major clouds, which is our focus right now. And open source is also taking contributions from the rest of the world. And all three of them are starting to show signs of life. In a way, open source is also architecture, believe it or not. That's the 10X architectural advantage.

Within that, branching is a byproduct of separating of storage and compute. We knew how important that scenario is. And since you're building your storage from scratch, you're building – the storage is all written at Neon. And we didn't really take any external components to that. Then you can architect it the way that branches are very, very cheap for you, because we use copy-on-write.

**[00:23:55] AD:** Interesting. So, the storage component of Neon is separate than like the standard Postgres storage component.

**[00:24:02] NS:** In a way. If you look at Postgres storage engine, it has multiple layers as well. And at the very lowest layer, Postgres requests pages from disk. So, you can like dive deep in the code and you will see that Postgres requests reads a page, 8 kilobyte-page, from a hard drive. And then there is a place where Postgres writes the transaction log record on disk at the time of transaction commits, and fsyncs that data. Postgres is obviously the system of record. You need to fsync to know its there.

We intercept these two code paths. And instead of reading a page from local disk, Postgres makes an RPC call into our storage and requested that page over the network. If you think about that might be slower, think about the fact that, oftentimes, people attach network, attach disks, such as EBS volumes, to run Postgres anyway. So, that interaction is going over the network anyway.

And the other one is writing a transaction log record on disk. Instead of writing it locally, we send it over the network into our service. That's called safe keepers. That's really it. So, the API between Postgres and storage are not your file system API. It's basically a custom-engineered API that fits very, very well into Postgres internals.

The rest of the machinery of the Postgres storage engine is the same. You know, heaps, B trees, indexes, all of that stuff stays. Vacuum, which is like an unfortunate byproduct of Postgres design is still there. But when Postgres write something to disk, instead of disk, it goes into the Neon storage.

**[00:25:42] AD:** Gotcha. So, just at that very boundary there. Okay, that's great. So, you mentioned the 10X architectural difference being separation of compute and storage. Like, what's the technological trigger that made this happen? Is it just the cloud and the elasticity of the cloud? What's different? Is it something about chips or SSDs? Or what's the big difference here that made this possible?

**[00:26:02] NS:** I think there are a few things that made it possible. I think it's like network latencies and VMEs. Again, network latencies and network bandwidth, they keep shrinking. And so, now, you don't really see that much of a difference between running Postgres locally versus having storage that runs over the network.

The second thing is – well, I think the scenario has matured as well. So, the scenario has matured, and some of the like was shown by Snowflake. And the scenarios of Snowflake are different, right? They're focused on analytics. But separation of storage and compute was a huge deal for Snowflake. And it unlocked certain scenarios.

In the meantime, the software development practices evolve their own CI/CD and Git. And now everybody needs preview environments. Everybody connects things, like Vercel, to their GitHub repos to pick things up and publish them on the web and create sandbox URLs that you can share around. So, multiplayer – I guess I shouldn't call it multiplayer. I should call it like just team collaboration. Went leaps and bounds. And those platforms took full advantage of this and provided a ton of value for that.

And so, database kind of felt stuck, right? So, there's a little bit of hardware and cloud enablers. But I think the biggest driver is what people actually wants to do with it. So, that allowed us to build this the right way.

**[00:27:30] AD:** I want to talk to you about just a few things you've been thinking about. And it relates to like a thesis I've been having. I've read like the Amazon Aurora paper, and the more recent Amazon DynamoDB paper. And like one thing you notice there with these AI cloud-based managed services is sort of the number of internal services it takes to manage all those different things and how they work together. And I was wondering if there was starting to tilt the playing field away from open source and more towards proprietary stuff, just because you can have these operational teams that can handle that.

But it's interesting, because you're open source. You have a few different services; compute storage. You mentioned Safe Keeper, or what the transaction log was. Is Kubernetes helping you manage that as well? Because now it's easier to run multiple services, rather than just being like, "Hey, I only want to spin up a Postgres binary and not run anything else." Whereas now, with Kubernetes, it's easier to run these systems. What's sort of enabling that?

**[00:28:19] NS:** I think it's a lot easier to put things together these days. You touched on a couple points. First of all, do clouds **[inaudible 00:28:26]** open source? We actually had a very interesting dinner with Christian Kleinerman, who is the Chief Product Officer of Snowflake. And his point of view, obviously from the Snowflake mountain, is, "Well, hey, it doesn't matter as much open source or not, because everything is cloud." And since everything is cloud, it's all about simplicity, ease of consumption, consumption-based model and value-driven to the customer.

And then on the other side of the spectrum, in the same category of data and analytics, there is Databricks, that started with the open source project, which is spark, and took full advantage of open source.

I think we'll see the world of both. I think every infrastructure project is going to be cloud. Full stop. And within every infrastructure projects that will be cloud that are fully proprietary, and there will be clouds that open source their implementations. Our mission is to build a fabric that runs the internet. And we want that fabric to be open source. I think it creates a more durable technology over time, and it creates opportunities to be absolutely pervasive.

And I think if you want to see an example of it, that's Postgres, right? That technology became absolutely pervasive. I don't believe into proprietary on-prem software. I think that category by itself, I think, is going away. I believe in cloud services. And kind of my dream is that open source should run the internet. So, that's kind of one of the reasons for us to choose that open source route.

**[00:30:00] AD:** Yeah. I love that dream. And just like never bet against open source. I also just think it's cool sort of the vision you're saying of having these read-only branches of your data may be stored on Neon's the technology. But then you can imagine Fly hosting like compute of Neon that connects those things. And you can have services sort of –

**[00:30:16] NS:** Yeah. Share a Slack channel with Fly. So, we're absolutely talking to Kurt about this.

**[00:30:21] AD:** So, in terms of branching, you mentioned copy-on-write, and how that enables it. What is copy-on-write?

**[00:30:27] NS:** Copy-on-write is when you request a particular page in the Neon storage. You refer to that page. Let's just simplify it a little bit. By page ID, and lock sequence number, and branch ID. And when you create a branch, all you do is create a new branch ID. And when you request a page in the new branch with a new branch ID, but it keeps the same LSN, it points to the same page.

Creating a branch is a size of metadata operation. I created a new branch IT. And then it forks it, right. And if you start writing in the new branch, it will start creating new pages that are not visible to the parent branch. But as long as you're querying old pages, which the majority – at least at the moment of the creation of the branch, the majority of the page pages will be all, you will be requesting the same page. So, you're not really doubling your storage when you creating your branch. That's what copy-on-write is.

When you start modifying that page, then we'll create an additional page. And the new branch will have that page. And the old branch will have the old page. But as long as you're not modifying anything, they will be pointing to the same branch. So, copy-on-write is a kind of general purpose. It's a concept. And our particular implementation does copy-on-write specifically at the page level.

What I knew about copy – the most popular open source to copy-on-write technology is probably ZFS. And what's interesting, there's a relatively big company built around ZFS and database workloads. And that company is Delphix. So, it's a company that packages ZFS and creates developer experience around those copies. And they use it to run all sorts of databases on it and improves Dev test staging environments in the old world, in the old on-prem world. I actually don't know. But the last time I checked, they were mostly proprietary technology that runs in the data center. So, I know their customers. I know their founders. I think it's a very cool idea. But it's due to the nature of their distribution, it's only accessible to a relatively small number of people and large enterprises that run fleets of databases on-prem. We're here saying, "Hey, this is a great idea. But we want to give it to everyone. And we want to give it to everyone for like trivial consumption in the cloud."

**[00:32:53] AD:** I want to talk a little bit about performance and like a few questions around like performance. How well you're trying to – how do you compare yourself to bog standard Postgres? Should it be pretty equivalent there. And especially as you move into things like offloading to S3, or sort of more dynamic compute, how does that affect things?

**[00:33:11] NS:** Yeah, this is a great question. Performance starts with goals, right? At SingleStore, to contrast this with Neon, our goals were to be the fastest analytical solution on the planet. The people who like absolutely love SingleStore really loves its performance. They're

like, “Well, this is the best thing, the fastest thing since sliced bread.” We sunk an enormous amount of engineering into this. And that what created this passion following.

At Neon, our performance bar is Aurora. So, we want to be just as good or better than Aurora. And we'll do what kind of what modern hardware allows us to do. And we certainly don't want to be worse than Aurora. And all our internal tests are comparing Neon with Aurora, especially with Aurora serverless. I think we may be pleasantly surprised. We are learning that Aurora is good, but not great. And that keeps us optimistic that we can be for certain scenarios. And databases are such a vast surface area that you can’t just say you're faster. You can be faster for this scenario. And that scenario will be relatively narrow.

We'll see certain scenarios were already faster than Aurora. For certain scenarios were slower than Aurora. And we're digging in. The way to work with performance is, well, understand all the variables that go into this. Be confident about the architecture. If you find issues, you have to change it. And then spend a lot of iterations of producing flame graphs and figuring out where the bottlenecks are. So, I don't have a better solution here. But that's the goals, is our goal is to be on par or faster than AWS Aurora.

**[00:34:48] AD:** Going back to your tweet earlier about finding an advantage, your second step was, “Hey, build a freakishly good engineering team for that.” What were you looking for the Neon team? How hard was it to find these people? I'm sure you've been in the database world for a while, but like what did that sort of look like in building out this initial team?

**[00:35:03] NS:** Yeah, when you think about this, you kind of want to start with the founding team. The founding team, in my opinion, should have all the ingredients, that given enough time, without any additional help, being able to build a full product, or at least build the full product in its first iteration.

And I'm not talking about the MVP, I'm talking about the product that I can actually generate revenue. So, there should be enough DNA in the founding team to be able for you to get there. This actually allows you – weirdly enough, allows you to hire better, right? Because likeminded people want to join the crusade.

And we're blessed with the fact that the two, our cofounders, Heikki and Stas, who are both Postgres hackers, possess all the required DNA to deliver on the vision. Well, the team is now pushing 40 people. And the majority of those folks are our engineers.

Here are the few things that helped us hire this team relatively quickly. And I'm very proud of all the capabilities of that team. Some of them are more controversial than others. And we'll talk about them. The first one is open source, right? Believe it or not, it's easier to hire for the open source project than for the non-open source project.

And then work is out there for everyone. On display for everybody to see. So, if you go and look at the Neon repo, very easy to see who's the most productive. By the way, that's Heikki. And then it's easy to see everyone's contribution. There are some really complex pieces of code that more like art. And then you see, and people are really proud of their work that they put into open source projects. And they know that others will be looking.

The second thing that helped us is Rust. We chose early on that our stories will be built in Rust. My friend, and the first engineer at MemSQL, Alex Skidanov, who later built a company, a crypto company called NEAR Protocol, told me once that there will be a day that somebody is going to start a database Project. And they will decide to build it in Rust. And they will move multiple times faster than an alternative project. And they're written in C++. And SingleStore is written C++.

And then I spent some time understanding why that is. And wrote small amount of code in Rust just like to see what it feels like. And then became clear that this is the future. And also, it's going to be easier to attract the next generation systems developer if your core project is written in Rust. That paid off. That paid off. One of the most productive folks on the team is a former member of Rust analyzer. And you know, it's just an example of just the choice of what you write your project in also contributes to your ability to hire.

The second is super controversial. The third one is super controversial. And then people just argue ad nauseam about this. And that is are you remote first? Or are you in the office in real life? And yesterday when we were going around the table, and there were some really cool companies in the dinner with Christian Kleinerman of Snowflake. And really cool, cool

companies around the table debated, “This is so cool when everybody is in one place. And if you're not, you lose on serendipitous interactions.” And then when you’re remote, you have easier access to talent.

So, we chose to be remote. We started during the pandemic. Our payroll started March 1st, ‘21. And we found it's easier to find just the right person if you're fishing in the global pond. But we're losing out on those serendipitous interactions. So, that's the third piece.

**[00:38:51] AD:** Yep, absolutely. It's a little bit of both. I was going to ask when you all started working on Neon? So, March of ‘21. So, about a year and a half, and you're already – I mean, it was a year and a quarter into public beta is all it took?

**[00:39:04] NS:** Correct? Yeah. And March 1st, we only had a – we had myself, Heikki, and Stas and a slide deck.

**[00:39:09] AD:** Wow! That's amazing. That's pretty quick. Two questions I want to get for you. Number one – So, you were a founder of an infrastructure startup before, SingleStore. Lots of great success there. I know some friends that really love it. What did you learn from that experience that you're bringing it over to Neon?

**[00:39:25] NS:** I think that thing that I learned, those are bits, right? So SingleStore is clearly success. Most likely, the company will go public. And that's really amazing. It's like how often this happens? But there were things that were hard. And SingleStore has a ton of technology. Meaning, if you say, “Oh, I've got to build a database. On top of it, you say, “I'm going to build all parts of that database. I'm going to build storage, and compute, and the cloud service, and various on-prem installs.” That is just a lot of technology. Within the database, storage compute, query optimization, query execution. Then you say I'm going to cover transactional and analytical workloads. So, it's just a vast system that people, when they look at it, they don't believe that it’s going to work. But it does.

At Neon, we're incredibly narrow. So, we're saying we're only cloud. Bits are open source. But we are offering consumption only in the cloud. If you want to roll the dice yourself, you're on

your own. It's only serverless. It's only Postgres. We're starting with only AWS. We're going to run other clouds.

And what I see that it allows you to do is it allows you to focus, right? And we also now have rebuild and compute. That's Postgres. We only build in storage. So, it's actually a lot less technology that we're building here at Neon that allows us to make that technology really deep. And it also allows us to create lots and lots of friends.

So, now, every analytical system is our friend, right? Because we are not encroaching on their territory. Every company that is building for the Postgres ecosystem. And we didn't fork Postgres. It's just Postgres. It's our friend. Every distribution platform that needs a database is our friend. Every organization that just runs Postgres already is our potential customer. So, that's one thing that I learned from the single store experience. And that's what we do differently. To think that we're going to do the same as the quality of the engineering team and engineering bar.

I think SingleStore ended up being a global company. There are at least seven offices, I think, around the world now. So, we're going to do the same from day zero. But in terms of like the quality of the systems engineer on the team is just as good a hire. That would be another lesson learned.

**[00:41:40] AD:** Cool. Just to wrap it up, let's talk about the future. What are you excited about? What are you seeing, either in terms of what's happening with neon? What's happening with databases? What's happening with applications? What are you excited about? And what are you seeing?

**[00:41:52] NS:** Great question. I think the problem of purely running your app on a cloud database is for the most part sold, right? And you can get that solution on the cloud providers. You can get that solution on Heroku is on the line, DigitalOcean, **[inaudible 00:42:06],** lots of lots of places. I think the problem of helping developers build an app on a database is far from sold. So, databases give you very, very little support for the modern developer workflow. And the architecture kind of prevents them from doing that, right? Which I think is the biggest opportunity for Neon.

Where the world is heading is increasingly rare model automation. So, once the system's part and the key workflow parts, like branchings, are out of the way, and all the integrations where the modern dev platforms are built, we're going to keep investing in ways of helping developers build their apps. Some of the things we're exploring, and I can't obviously commit to any of that, because it's in a lab. It may or may not happen and may or may not be the right approach. And sometimes I say something and the team freaks out. But the stuff that we are looking at is, with the advances of AI, it seems silly that we, as developers, still need to choose indexes for our databases. It feels like this should be automated. The enabling technology at Neon is one way creating a test playground. So, we can create a test playground and auto create indexes. And then we know your workload as well. So, we can prove to you that your workload – that performance improves, and you don't break anything.

Does it have to be AI? Maybe. It doesn't have to be. There are like research started from like early 2000s that can do that just imperatively or like regular algorithms. Or we can do it with AI. I don't know what the right solution is. But I do know that it's kind of silly to push that complexity on to users.

Speaking of AI, though, think about a workflow of a modern developer. Usually, this person is an expert in whatever language this person uses. Should it be JavaScript, or Go, or Java, or whatever. It doesn't matter what it is. But that is the primary programming language for this person.

And typically, with the interaction with a database, that language is – SQL is not the number one language for that person. And so, that's where like Copilot, or Replit, GhostWriter could provide additional value to go and help you write that SQL. And Neon can be a part of that, or Copilot can start generating SQL, or interaction between the two.

The other thing is – and those demos keep blowing my mind. You can generate code or you can translate from one code to another. And OpenAI famously showed how to translate from JavaScript to Python. And when I look at it, I don't understand why we wouldn't be able to translate from JavaScript to SQL and make it a part of a workflow. Of course, you need to know what the schema is, but that's where Neon comes in, providing that information to AI.

Once you are a lot more fluid translating code or generating code for your program, you're not interrupting developer flow. And that's a big deal. And then from there, there's all sorts of databases that are out there. There are companies that use Sybase, and they have – Sybase is – It's not long gone. It's still there. But like, nobody wakes up and starts new projects on Sybase. But they have this like large footprints. And then those have stored procedures and whatnot. So, it would be great if we could just migrate those stored procedures using AI, using like code translation technologies into Postgres, and provide some sort of interactive way for developers to do that. These observations is like people didn't know SQL. But like a small subset of those people know Postgres stored procedure language. And so, that's where like generative technologies, that's where autocompletes, that's where Copilot can make a big difference in just how fast you progress.

**[00:46:01] AD:** Yep. Awesome. And especially like – I know, in the JavaScript, TypeScript ecosystem, you're seeing things like Prisma. So, you get more typing around your database, and that taking off. And that interacting with copilot and being able to, like you're saying, go from JavaScript. What you want JavaScript and getting it directly in SQL in a nice, clean way. That's pretty awesome. I assume that your team is doing something at the storage level a little more involved than just copiloting from C to Rust for the Postgres storage.

**[00:46:27] NS:** Oh, yeah, yeah, yeah, yeah, yeah. We don't do any of that. Yeah. The storage system is built from first principles and from scratch. You can trace it, and it's all public, right? It's all on GitHub.

**[00:46:36] AD:** I'm sure that's some cool stuff. And I'm excited to see the progress here. Again, I started with Postgres. That was my first database love. And I'm really been looking for something that fits well in the serverless world. So, Nikita, thank you for coming on today. And I'll be watching Neon going forward. For people that want to go check out more about Neon, where can they find Neon? Where can they find you?

**[00:46:55] NS:** Well, there's neon.tech. My email is nikita@neon.tech. My Twitter handle is Nikitabase. I keep working in database technologies. And so, I think it kind of fits. Yeah, those are the two. DMs are open.

**[00:47:07] AD:** Awesome. Sounds great. Nikita Shamgunov, thank you for joining us on Software Engineering Daily.

[END]