EPISODE 1169

[INTRODUCTION]

[00:00:00] JM: Internal tools are often built with Ruby on Rails or Node.js. Developers create entire full-fledged applications in order to suit simple needs such as database lookups, dashboarding and product refunds. This internal tooling creates a drain on engineering resources. Retool is a low-code platform for creating internal tools. These internal tools can be written by biz ops people, marketing people or other roles other than engineering.

David Hsu is the founder of retool and joins the show to talk through what he's built.

[INTERVIEW]

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

[00:00:40] DH: Thanks. How's it going?

[00:00:41] JM: All right. Retool is your company. Retool is for building internal tools. Explain what an internal tool is.

[00:00:49] DH: Yeah. Internal tools are sort of all the tools that software engineers build for other people in the company. So sales, marketing, support, operations, stuff like that. So for example, if you want to be managing all the podcasts, adding tags to particular podcasts, stuff like that, that's probably some internal tool or some CMS that you have that allows you to do that.

[00:01:11] JM: How have these internal tools historically been built?

[]00:01:16 DH: So typically software engineers basically build stuff from scratch using some language like JavaScript and probably some framework like React, Angular, Vue or whatever else. And this is actually pretty inefficient. It's actually how we started Retool actually, was we

ourselves built a lot of these internal tools. And after you build a good amount of these, you realize they all do very different things. So some of our customers have tools, let's say, to process withdrawals. Some customers have tools to manage promo codes. Some customers have tools to manage menus for restaurants, a lot of stuff. These all do very different things.

Fundamentally, they actually all have kind of the same building blocks. Probably your tools are not too different either. It's probably just a bunch of tables, a bunch of buttons, a bunch of drop downs, forms, text inputs, that kind of stuff. And so after we built a lot of these internal tools from scratch every time, we ourselves realized, “Hey, maybe there could be a faster way of building all of this stuff. Instead of building it from scratch writing React or JavaScript every single time, what if there could be a sort of much higher level way of building this instead of, let's say, when you drag on a button, when you want to, let's say, build a button, you have to worry about debouncing the button. You have to worry about firing the post request. You have to worry about showing errors when it fails. And that's actually a lot of code to actually write.

And so all these tools, the way you build them is you build them from scratch, right? You sort of write JavaScript probably using a framework like React, and it is quite slow to build all this stuff from scratch and it's quite tedious. And so when we ourselves built all these tools over and over again, we were thinking to ourselves maybe there's a faster way of building all this stuff. What if there are higher level way of building all the stuff? So that's basically Retool. So in Retool you use a drag and drop interface to build something like 50% or 60% percent of what you're looking to build. And then the last 30% or 40% is then built – You write a little bit of code to sort of customize exactly what you want.

[00:03:15] JM: Why do you start working on retool?

[00:03:17] DH: Sure. Fundamentally, I think good engineers are to some extent lazy, because they don't like doing the same thing over and over again. And so in our case, we ourselves have built so many internal tools before that eventually after you build 10, 20, 30 of them you realize there should be a faster way of building this stuff. And so the exact inspiration point actually was when we started retool we were looking – We were sort of building internal tools

to do a lot of things. We were starting a fintech company at the time, and some things you have to do include, for example, when someone tries to withdraw money, you have to verify who they say they are. So we do some KYC, you have to do AML. KYC is know your customer laws. AML anti-money laundering. There's a lot of fraud as well when people are sending money back and forth.

And so for all these tools what we would do is we'd actually build maybe the first half or so in some sort of open source BI tool, probably something like Metabase, maybe Tableau, Chartio, etc. There's 30 of these BI tools. But then when we actually want to take action on that data, so let's say we look at the withdrawals of Metabase, we say select star for withdrawals, order by id descending or something. So we see all withdrawals. But then when we actually want to go and approve them, there is no BI tool that allows you to actually write data back to the database. If you sort of look at the 30, 40, 50 BI tools, none of them support that.

And so when we had to do that is say, “Okay, well, now we can see the withdrawals. Now we actually have to go build a tool or go use the console.” And so at the beginning what I would do is I would open a Rails console and then go find it's like w equals withdrawal dot find open parent 42 close parenthesis or something, w dot approve xclam to make it actually work. But of course this is really inefficient, right? Like I can use the console, but when we started hiring people, we certainly did not want to give people console access. And so then we actually had to go create sort of bespoke internal frontends built. I think we actually built an enclosure script for fun. But I always had to sort of build these frontends from scratch basically just to write back to the database.

And so to us that seemed really stupid, because these BI tools are sort of very good for reading data from your database. But as soon as you want to write data, you have to start entirely over from scratch. And so that was to some extent the inspiration where, “Hey, all these BI tools are very good for sort of –” If you, let's say, you want to chart one table, you can write SQL get table get chart. It's already commoditized. But there's nothing similar for writing back to a database or to an API. That's what we'd call sort of internal tools original applications basically.

[00:06:01] JM: You first started working on it in 2017. Tell me about the initial version of Retool.

[00:06:07] DH: Yeah. The initial version was very basic. When we first started, I think we basically supported one workflow, which actually turns out to be a very common workflow. But basically sort of one workflow, which is you want to see a particular table, let's say users, for example. And then you want to be able to search those users. Do something against the users. Maybe change a boolean field or something. And then see another table that has a foreign key somehow connected to users. So maybe it's users and transactions, for example. Or users and purchases or something like that. And so that was fundamentally the use case that we solved, sort of crud on top of two sort of semi-interconnected tables basically. And then we just started showing that to customers, and surprisingly that is actually a pretty common workflow. Sort of seeing two linked tables and doing something, searching would be the sort of first one, taking action the first one. Second one doing actually the second one. And so that's where we started.

And then, as you can imagine, customers started asking for a large amount of things. I think what's interesting about Retool is that it's almost like a new IDE or almost like a new programming language. It's a new way of building applications, right? It's fundamentally quite different from using React or whatever else. It's a visual way of building all these stuff. And so the challenges we face are quite broad. So in particular, for example, I'd give you like a small example feature request that became quite big or that became sort of quite important for us was when we were working on the table component, the sort of amount of requests for modifications or sort of advanced customization of table components kind of mind-boggling. People would ask, for example, "Hey, can I color particular columns? Can I sort by multiple columns? Can I actually map the columns themselves?" So instead of showing, let's say, the ID, I want to show the ID times 10, for example. And so we had to sort of find ways to support all these things, know multiplying ID by 10 is not a reasonable request, right? And so I think the interesting part about Retool is that it is sort of so broad. It's like building a new programming language. So you have to support a lot of things. Retool is sort of very broad as a product. And

so if you think about all the things you have to support, if you think about a programming language, a programming language has to allow you to do practically anything. And if it doesn't or if it's uncomfortable to do something, I've actually changed the program language to allowing you to do it better.

So the interesting thing about Retool is that the work to be done really is quite broad and it's quite interesting because of that. I think if you had told me that you're to work on building internal tools for the rest of your life, I think that's interesting. But telling me that I get to work on a new programming language or a brand new way of building applications, I think that is really, really quite interesting.

[00:09:02] JM: Tell me about the stack that you used to initially build Retool.

[00:09:06] DH: So we got quite lucky uh in the sense that we started at quite a good time. So the frontend the backend are pretty simple. So the frontend is JavaScript. And we might go over to Typescript maybe a year ago now. I think we're 90% of the way there now. We just found a lot of bugs that we could have called types basically. So anyways, JS type script on the frontend. And on the backend mostly JavaScript today, maybe 50% of the migration there to Typescript and node in the backend.

The part where we are really lucky though is that we ourselves deploy via Docker and Kubernetes and that has actually enabled us to release on-prem versions of Retool that are just Docker images that are quite simple. And so now today a large amount of our larger customers actually run Retool on-prem. I think that's interesting for a variety of reasons. The reason they run it on-prem is because they have security considerations or maybe they have, let's say, continuity considerations. And so when you're running Retool on-prem, no matter what happens to Retool as a company, you will continue to have Retool running forever sort of in your own infrastructure.

And all this on-prem stuff is only possible because we started it around mid-2017 when Docker and Kubernetes became quite popular. And so for us, to deliver on-prem versions of Retool,

it's exactly how we host Retool ourselves. And so it's really quite simple. Had we started three years before, I think it would have been substantially harder. It would have been via Chef, via Puppet or whatever else. And supporting on-prem would have been a whole different ball game.

[00:10:38] JM: So given that the stack was pretty a self-fulfilling prophecy, what were the difficulties in getting the first version of Retool working?

[00:10:47] DH: Oh yeah, I think there were a lot of challenges from a product and from a technical perspective. So from a product perspective, going back to this analogy, it's kind of like a new programming language, right? And so a lot of things we'd have to consider, it's like do we want to build that into Retool or not. And to what extent do we want to support it or do you want people to hack it? So let me think of a good example here. Maybe a sort of small example is Retool supports connecting to any REST API or even any GRPC or GraphQL API. But for some very popular services such as, let's say, Stripe, we actually have a built-in connector that you can use where it sort of shows the documentation for every endpoint that you could potentially be hitting. Tells you what parameters you need. We basically generate it from the OpenAPI or Swagger spec basically.

And so this is a small thing, but to put to consideration is what APIs do we support natively versus what APIs do we just let you sort of hit via REST or GraphQl or GRPC basically? And that's one small example, but there are more advanced examples such as tables. In tables, we could allow you to specify which color to color every cell. We'll give you like a drop down or something. But we decided not to do that. We decided instead we just let you write JavaScript and sort of decide what color the cell should be. And so that way you can choose any hex color you want and you can even in fact even make conditionals. You can say, "Hey, if let's say the cell value." And so you can use JavaScript to customize the color of every cell. And so you can say, "Hey, if this cell value mod 2=0? Then it should be green. Otherwise it should be red."

And so that's an example of something that maybe it's slightly harder for a non-technical user to grasp, but dramatically increases the flexibility of Retool itself. And so that's an example of

interesting product decision. A similar thing is if you're, let's say, working on a language, let's say, like Ruby or JavaScript or whatever else, you have to think a little bit about whether you want to add, let's say, new features into JavaScript, for example. For example, ES has, for example, constants, right? And so that's a feature that it's similar to sort of thinking through is that good, is that bad? What does that enable? What sort of downsides are there? That's sort of the kind of stuff we think about for a product perspective from building Retool.

From a technical perspective, a lot of the challenges have to do with scaling and security. So for example, on the security side, the fact that you can write JavaScript and Retool is tremendously powerful, but actually also tremendously scary, because in some cases, for many of our customers, our customers may not even trust their own employees actually. And so we actually isolated the JavaScript and we run it in a sandbox. And if you don't do that, an employee or a creator of a Retool app could actually exfiltrate from the Retool end user their credentials or their cookies and stuff like that and then send them back to sort of any endpoint they wanted to. And the fact that we sandbox is prevents that.

And so there are a lot of interesting considerations over there on the security side. On the performance side, as you can imagine, because we're connecting the databases and APIs and potentially piping lots of data through our back ends, there's a lot of stuff we worry about there as well. In Retool today, I think we have a lot of work to be doing on the performance side certainly.

[00:14:10] JM: All right. Well let's go through an example of a modern internal tool I might want. So we have a Google Spreadsheet at Software Engineering Daily that manages all the podcast guests that are coming up. We have their email addresses, and let's just say I want to send an email to all of my podcast guests, what would be an experience of doing that?

[00:14:31] DH: Yes. Backing up very briefly, I think what's interesting if you think about Retool is sort of – And which use cases is good? And so a good example is exactly the sort of guests coming up. If it's stuff that you enter manually by hand, oftentimes Google Sheet is sort of well good enough for what you're looking to do. If you have 50, 100 rows, it's pretty easy to

manage that. And so if you have something like 50, 100 rows of manually entered data, a Google Sheet is actually pretty good for that. It's pretty straightforward to manually enter the data and you don't have to worry too much. You don't have that many advanced UIS on top of this or you don't need such advanced UIs on top of it.

However if let's say SE Daily has been going for a decent amount of time now and you guys have a lot of episodes. I've been a listener. I'm guessing this stuff is sort of a database somewhere where you have sort of all the previous guests. You're like, “Okay, this guest hasn't been around for maybe two years. It's time to check back in on them. Maybe bring them back on again if you get an update and sort of how things are going, etc.” If that stuff is stored in a database, then the only way to build an application on top of the database is really writing code. And that's typically writing React or writing JavaScript and using React.

And so where Retool sort of comes in is if you're really looking to build a UI and the sort of comparison or the sort of the best next option is React. Then Retool is quite good for that. If you're just managing a few rows inside of Google Sheet, probably Google Sheet is actually better for that. But anyways, to go back to your original question now, we do support Google Sheets as a backend. And so you can actually connect Retool to Google Sheets. And so what you can do is launch Retool. You basically log in via Google. Choose sort of the files you want us to give us access to once you select the spreadsheet. And then what you can do is you can start building UIs on top of that very quickly. So you can, for example, drag on a button. Maybe you want to drag on a table first to show all the rows in the Google Sheet. And you can enable multi-select on the table. So you can actually go select multiple guests that you want to email or something or you want to select all, select all of them, and then you might want to drag on a button that says, “Hey, you can configure the button on the right hand side with a bunch of properties.” And so you can say, “Hey, this button should actually go make a post request, let's say, to MailChimp, or Sendgrid,” or whatever your email sending service is. And then you can sort of design your email and send through MailChip or whatever else and say, “Hey, when I press this button, I want to send the post request and actually send that email.” And so it's fairly simple. It would take maybe two, three minutes to build this. Whereas uh if you build it from scratch fire React, it would probably take substantially longer. You probably have to worry

a lot about oauthing into Google Sheets, which itself is – The Google Sheets API is quite difficult to work with. But then also sort of manage your credentials, MailChimp, etc., stuff like that. So building it from scratch prescriber may take you a few hours and Retool will probably take maybe five, six minutes, something like that. So substantially faster.

[00:17:29] JM: Ok. So if that's a simple example, let's go through a complex example. Tell me about a very complicated app.

[00:17:34] DH: Yeah. I think a good example is if you think about very complex workflows, and these are primarily in sort of larger companies. So let's say that you are, for example, Plaid. Plaid is a customer of ours. And Plaid at this one is a fairly large company. We have a lot of things going on inside there. And a lot of things are very sensitive, right? Let's say someone is trying to get access to a bank account. So let's say you take a customer like Plaid, for example. Plaid is a fairly large company now. They have a lot of things going on internally, and they have sort of various levels of support reps. And so they have a level one support rep that if you write in and you say, “Hey, for some reason, I try to get the balance for this particular customer's bank account and it returns. The number is actually wrong,” let's say. You're a developer, you're just testing it. You’re like, “Hey, I'm testing it on main account.” For some reason the balance is wrong. Plaid probably wants to go build applications that allows Plaid themselves to go call their own APIs or to see your – Just go test their own API and see sort of transactions or the balances in a particular bank account.

However, they probably don't want, let's say, a level two or level three support reps to do that. Instead they may only allow level six or level seven sport reps to do that. And so you can imagine there are sort of a lot of these sort of more restricted or more sensitive, let's say, applications that a company may build. And they only want to show these applications or these capabilities to, let's say, particular sets of people inside of the company. Or maybe when you sort of take an action on a particular user, let's say, Plaid in this case, maybe let's say creating new promo code, for example, that saves the user a substantial amount of money in their contract. In order to do that, potentially, you want to go through sort of various levels of approval.

So maybe the account executive themselves can't do that. They actually need to escalate to the head of sales who can then approve discounts up to 20%. And then you can go to the VP of sales who can approve discounts about, let's say, 50% or something like that. And so these are examples of applications that start becoming quite complex in terms of sort of the operations around them or who can use them? Who needs to approve every action? That kind of stuff. And that kind of stuff, if you try to build that in-house, it'd actually be quite difficult. You'd have to basically build some sort of voting or consensus system where you say, "Hey, for particular API endpoints, we want to actually go ping three people to sort of get their approval on them,, or for withdrawals, or let's say discounts about 50%. We actually want to get three people from this level or seven people from this other level and whichever one comes in earliest is fine basically."

And building that from scratch it's actually quite difficult, right? Whereas in retool, it's basically supported out of the box. And so you can say, "Hey, this particular query that turns out a post request to make a new discount actually should be protected and it should be protected and it requires approval from these groups that we can actually sync to Octa or any sort of single sign-on provider." And you can say, "Hey, it requires three approvals in this group, or five from this group, or seven from this group or whatever else." So that's an example I would say of a more complicated application. That would be very difficult to build in-house. You'd probably have to just go write a lot of code in your backend to go support this. But in Retool we support directly out of the box, which makes it substantially faster.

[00:21:06] JM: How do you use Retool at your own company? How do you use Retool at Retool itself?

[00:21:11] DH: Yeah. So as you can imagine, we have a good amount of internal tools that we build ourselves. For example, when we manage and on-prem deployments, on-prem deployments have a license key. And so we create license keys by a Retool app that we build ourselves. We manage a lot of the analytics stuff as well. So a particular user writes in and says, let's say, "Hey, my app is broken," for example, "and I don't know what caused it, but

could you please revert back." This is for non-on-prem customers. For cloud customers, "Could you please restore it back," let's say, "three days or something. It's easy for us to do via Retool as well. So primarily I would say for support in operational use cases.

Actually a pretty interesting use case too is on the sales and engineering side. On the sales side, I don't know if you've ever tried using Salesforce. It's quite slow. It sort of I have not never met anyone who really enjoys using Salesforce, but everyone uses it obviously. So any sort of legitimate SaaS company uses Salesforce. So we use it ourselves. But we've actually built our own frontends on top of Salesforce to let our sales engineers enter data into Salesforce better. Because Salesforce, if you sort of go to let's say the opportunity field, which is how you manage the opportunities sort of any given person is working on, it's actually quite difficult to enter data into there. It doesn't have very much context. It shows you all the fields whereas you only want to edit one or two of them. And so this is quite bulky, basically. We ourselves actually built an application on top of Salesforce, basically on frontend and top of Salesforce that makes it substantially, that makes Salesforce substantially nicer to use. So that's an interesting example as well on the sales side.

On the engineering side, I think this is actually really quite interesting, is there is this internal tool that Stripe themselves built not using Retool that basically allows engineers to every day wake up and see sort of where their PRs are at. And so it's built on top of GitHub and it basically shows you, "Hey, of the PRs that, where you are the reviewer, which ones are blocked on you? Which ones are ready to merge? As well as for the PRs you have out, which ones are waiting on review? So you should go ping people to make them review your PR as well as which ones are also ready to merge. And this is an example of an app that if you go to sort of four separate screens in GitHub, you can go find this data. But GitHub actually does not have a sort of single view of all of this.

And so what Stripe did is they actually built an internal tool that allows their engineers to sort of get a sort of bird's eye view of all the PRs they have sort of waiting on them or blocking others so they can sort of move their PRs forward. And so this is an example of a Retool app that we've built in-house, but we actually also turned into a template. And so you can actually

directly go to retool.com templates and clone this application and then just log in via GitHub and you can just go use this app yourself. And so I think that's sort of an interesting side effect of Retool, is by sort of commoditizing internal tools across different companies, we can sort of enable all companies to substantially up-level their quality of internal tools as well as have internal tools that they probably would not have built themselves know to substantially up-level their internal tools.

So we, for example, as a retool app, as a sort of 10% or 15% engineering team, would probably not have spent the effort to go build an internal tool that allows our 15 engineers to go track GitHub PRs. But because it's a template to Retool that now we can just directly go and clone, we get sort of the internal tools that Stripe has without having to build them in-house. I think that's an interesting side effect of a Retool.

[00:24:49] JM: Retool is a very detailed application. There're lots of data sources, UI components, APIs to connect to. there's this big end-by-end matrix of possibilities of apps that people could build. And if you want to satisfy this end-by-end matrix of all the different data sources and APIs and different ways that people could build apps, you've got a lot of ground to cover. When you started Retool, was there a specific subset of those components that you were focused on?

[00:25:21] DH: Yes. So when we started Retool, we mostly focused on data entry or data editing into databases and APIs. And so these sort of core two resources, we call them sort of backends that we connected to, or Postgres as a database. So we could both read and write from Postgres as well as REST APIs. And so you can make get requests, put requests, post requests, etc. So those were sort of the two main data sources we supported. On the component side, we basically supported text inputs, drop downs, sort of most things you'd find in bootstrap. Maybe the top 10 or 20 components we find in bootstrap.

What I think is interesting is if you think about Retool, it is quite broad. But fundamentally a programming language itself is also quite broad, right? If you think about, let's say, Javascript, you can get a computer or a web browser to do practically anything with JavaScript, right?

And Retool is similar in that regard, sort of the core language or sort of core building blocks of Retool are actually quite compact. It is just that the – Because the building blocks are so primitive, or I mean we improved is not the right word. I think JavaScript is quite primitive, I mean, Retool will be higher level. But nevertheless, the building blocks are quite un-opinionated I would say. The table is not just for, let's say, displaying orders, right? The table can do anything. It's a stable component. And so because the components themselves are quite un-opinionated, you are actually able to build quite complex things with sort of the building blocks that we give you.

[00:26:51] JM: Tell me more about how retool actually works, how the editor works. So when I load up a new version of the editor to build a new internal tool, what's happening?

[00:27:00] DH: Yeah. So how it works from a technical perspective is all retail applications are basically JSON blobs. And so if you, let's say, build an application where you drag on let's say a table, a button and write two queries, right? So the table maybe pulls in data from, let's say, Salesforce or Postgres or whatever else. So you might write select star from opportunities, right? That's a Salesforce query that pulls in data from Salesforce. That, as you can imagine, is some sort of JSON where the query is select star from opportunities, but query type is Salesforce and there's a bunch of other properties such as you know query timeout. Caching is probably not turned off by default, all that kind of stuff. That's just JSON blob basically. And the table, then you can connect that to that query by saying, “Hey, this table should pull in data from query1.data or salesforcequery.data or something.” And so it pulls in data from that Salesfoce query. And again the table just has a bunch of properties. It's kind of like React, right?

If you think about React, if you had a table, you probably have let's say the column widths. You might have, let's say, the data. The data in this case would be a salesforcequery.data. You may have let's say a column ordering, stuff like that. So we sort of have all those props, and that all is just JSON. And so a Retool application is basically just a bunch of these components and a bunch of these queries. And there are some other things too. But fundamentally, it's just a giant JSON blob basically. And so when you first open the Retool editor, it would basically load

the JSON blob from our backend and then we render it on the frontend, and the front end basically takes the JSON blob, parses it and then renders into tables, queries, buttons, all that kind of interesting stuff basically.

And then when you start modifying it, it's right back directly to these sort of APIs to say, “Hey, you added a table at position three-five. This table is of name, Salesforce data, etc.” And so that's essentially how it works. And then the interesting thing is because we actually – Because it's actually all JSON blob, you can actually do a lot of things that sort of most I would say sort of low-code or no-code tools would not support. You can actually sync the stuff all to Git. And so I think what's really quite interesting about the fact that it's all JSON blob is actually we actually let you serialize this JSON blob into YAML, which then becomes quite readable, and then sync it to Git. So you can actually do version control on top of your Retool applications. And this is really interesting because it means that you can start doing PRs.

So for example, you can say, “Hey, I'm an engineer building applications, but I actually want to let my support team build simple applications too. And so they can go build these applications, but maybe they can only commit to, let's say, the dev branch. And then they have to request a PR or review. And then I can then as an engineer review the PR and then merge it into master.” And so you can sort of get the benefits of writing code. I think sort of a lot of the benefits of writing code are ancillary to sort of writing the code itself. It’s really sort of the version control, the testing and everything. And so we support doing all of that even though we let you build the application substantially faster in the drag-and-drop way.

[00:30:08] JM: If I build a really detailed complex app, are there performance limitations that I'm going to hit?

[00:30:16] DH: Great question. So mostly performance limitations have to do with the amount of data you're piping into Retool or sort of directly manipulating inside of Retool. So if you think about, let's say, you know you have a database that has, let's say, 20 million rows, let's say, and maybe, let's say, 10 million columns, for example, quite large. If you tried writing select star from big table, you would pull in a lot of data. And so the performance limitations are both

on the frontend and the backend. So on the backend, what our backend does it actually sort of connects their database and tries to pull the data. And if the data blob is, let's say, probably be like terabytes or something, that really might take a while. So we don't recommend doing that. And so we recommend using some sort of pagination to pull in, let's say, a limited amount of data. If you're pulling in, let say, a thousand or even ten thousand rows times like 10 or 20 columns, that's no problem.

And then once data is pulled into your browser on your frontend, you then have to worry about performance limitations that Chrome will run into. And so you only have so much RAM on your computer. And so if you tried loading in sort of 10 million row table with 10 million columns, your browser probably will not have a good time with that either. And so the limitations really are with either sort of your database or with your frontend. Retool itself doesn't really do that much to your data. I mean you think about sort of building a web application, right? Let's say you build a simple web app where you are making a Git request that returns a giant blob of data and you sort of put into a table. Really, the performance limitations are regarded the network request, how long does it take for your backend to fulfill that and to send the data to your browser, as well as can your browser handle the amount of data that is in it?

So, really, those are the two limitations. When you're building the applications, there are some performance problems when you have lots of data in a table, because when you actually, for example, drag and drop the table, we re-render the table, right? Because we're using React. And so that does get costly if you have a lot of data in your table. But really when you're using the applications themselves, which is really when you really care about performance, it really is how much RAM does a computer have as well as can your backend return the data performantly?

[00:32:28] JM: Tell me about the cloud services that you use.

[00:32:33] DH: This is funny. So we actually use all three. So we use AWS, we use Azure and we use GCP. So we use them mostly for different things. Really, most of our stuff is on Azure today. The reason we're on Azure is because when we first started Retool, Azure gave us $500

of Azure credits. And so we started on Azure and have been on them ever since. That said, there are a few Azure services we don't use because they're fairly immature. So for example, RDS, we still use Amazon. So when you log into Retool for the first time, we create you a sample kind of test database or scratchpad database that you can use to store some data or just for testing purposes. And that is an RDS, because Microsoft – I forgot what their sort of equivalent RDS is. Does not have great Postgres support. And so we use RDS for that reason. And then we also use – For GCP, we use BigQuery mostly for analytics.

[00:33:35] JM: You said you manage your own Kubernetes, right?

[00:33:38] DH: We do, yeah.

[00:33:39] JM: Why not use container instances or ECS or whatever?

[00:33:45] DH: From our perspective, the compute of Retool is not really the limiting factor. Even today, if you look at sort of Retool cloud, we have a lot of customers that are really using Retool quite a bit. But even still, Azure bills are actually quite minimal. And the reason is if you sort of go back to performance discussion, really, most of the performance or mostly hard work is done by the database or the API. And so if you think about our backend that connects to a database, really, it's just varying data from your database to your frontend. And so it's just sort of piping data through basically. And so the performance implications, it really doesn't require that much compute basically.

More perspective, given that we don't require that much compute and we're not scaling too insane amounts or insane clusters or whatever else, it's easier to manage it yourselves. That said, in the future, we certainly want to go multi-region. So as we go to multi-region, it's something that we would certainly look into or explore.

[00:34:44] JM: I was speaking more about the difficulty of managing Kubernetes versus managing, for example, these Azure compute instances or Azure container instances or

Amazon elastic containers or whatever container instances they have, basically using standalone containers instead of Kubernetes.

[00:35:02] DH: Yeah. So this may just be a relic of sort of when we started. I think when we started, when we first looked at this, it was relatively immature. This was maybe three and a half years ago now. I'm sure it has evolved quite a bit since, and I'm sure we should probably look into it at this point. It has not been our top priority so far, because everything is working so well on the DevOps server side.

[00:35:26] JM: What's the life cycle of a database query for one of my Retool apps?

[00:35:32] DH: Yeah. So when you run the database query, so let's say you open the Retool app, right? When you open the Retool app, that will – Let's say if a query that runs on sort of application load, what happens is your frontend will then make a request to our backend to say, “Hey, could you please run query one?” We also do a lot of interesting things here to prevent SQL injection, for example. So we use parameter statements, etc. So let's say you have – You're query has select star from users, where first name I like curly-curly, text block one has value. We then also pass in the parameters into the backend as well. So we say, “Hey, I'll run query one.” And these are, let's say, the parameters that I want to pass in.

So for the database query, the first parameter would say Josh or something like that. And so you pass that into our backend. Our backend says, “Okay, cool. You want to run query one and you want to run query one of these parameters.” And so we basically then connect to your database and say, or we first look at query, we'll connect our database. See what query one is. And we say,”Okay, query one looks like it is on this particular resource. It's a Postgres resource.” And so we're going to go to our Postgres connector. And then that has a pool connections that connects to a bunch of different databases. Maybe sort of connected or maybe not. And so if it's not, it will connect to your database and then go run that query. Your database will go run that query. It’ll return the results to our backend. And then our backend will then return those results then directly back to your frontend, which is your Chrome browser.

We do some logistical stuff in the middle as well. So we have audit logs, for example, that we manage. And so we say, “Hey, looks like this user ran the Git user’s query on the Postgres database. It took this many seconds. They ran it from this IP address.” All that kind of fun stuff. So there's some notes that we take. But fundamentally, it's pretty simple.

[00:37:22] JM: Go you have a set of management principles that you use for running Retool that's distinctive?

[00:37:28] DH: Yeah. So I think what has changed a decent amount since starting Retool to now, for me at least, is at the very start of a company, the best way to move the company forward is by doing the work yourself. You're really just pushing the rock up the hill basically and you're sort of furiously pushing it. But then as the company gets bigger, at five people or ten people, the impact that we all have of the company diminishes to some extent, right? Today at Retool, 40-ish people, everyone sort of – It's 2.5% of Retool basically. And so of course the work we do is tremendously important to Retool obviously. But in the end, sort of the impact goes down to some extent. But the impact that we have on the rest of Retool goes up.

And so I think one change for me since starting Retool and having been here for a while now has been that the sort of impact that I have really is on the people around me and how can I make them more effective. And so for me, today, I'm the CEO, right? And so on average let's say 40 people to 100% makes 2.5 decisions. Maybe I'm the CEO. So maybe I'm making a 4% or 5%. So maybe sort of slightly higher than average. But nevertheless, I only make 5% of the decisions at Retool, right? So 95% decisions are made by other people.

And so really my job is not to make decisions, rather it is to empower other people to make the correct decision. And a lot of that is giving the context. A lot of those hiring the right people that have a similar values to the rest of us. And so a lot of my job today is really how can I enable other people to make the correct decision rather than making the credits for myself? And so that I think is one management principle that I think is quite interesting and something

that I've learned since starting Retool. And so similar themes are I think empowerment is quite important to us. And so if you look at retool out of 4dh 40-ish people today, roughly a third of us used to be former founders. We've started our own sort of companies, projects or whatever else. And that I think is quite important to us too, because we want people who are self-starters who really want to go have a big impact, but just need to be sort of pointed in the right direction or in the sort of right vague direction.

And so from a management principles perspective, I think a big theme for me really is empowering people and hiring the right people, hiring motivated people and sort of giving them sort of high-level principles about where we should be going and then setting them free. Let them do their best work.

[00:39:54] JM: What's your long-term vision for the company?

[00:39:57] DH: The goal for Retool is to be the way engineers build software. We're starting with internal software today. If you think about – Going back to your original question, if you think about sort of how software's built, it's actually quite surprising how inefficient it is to build a simple application that writes data back to a database or an API takes a surprisingly long amount of time. And a lot of the building blocks really should be pretty similar, but it really should be a much higher way of building these apps. If you think about building simple applications today, versus 10 years ago, versus 20 years ago, 20 years ago, maybe you would use Java Swing, for example. You probably write maybe a thousand lines of code to write some data back to a database, right? Maybe 10 years ago you'd be using JQuery, let's say, JavaScript. Today you'd probably be using React to Javascript.

Fundamentally, programming hasn't really changed in any sort of fundamental way in the past 10 or 20 years. And so Retool is trying to change that. We're trying to be sort of a substantially or fundamentally new way of building applications. And the hope is today Retool only built internal frontends. The hope is eventually to expand on sort of other internal applications including backends, including scheduled jobs, like cron jobs, for example, including more work-flowy type things. So let's say if this happens, then do this. And that connects to the

database and then sort of take a look inside there. And then if this is true, then do this, etc. So we want to get to the point in the next few years where we are the way developers build internal software. And then eventually one day the way developers build all software. But that's still quite far away.

[00:41:34] JM: It's pretty ambitious. You studied both philosophy and computer science. What have you found at the intersection of those two disciplines?

[00:41:43] DH: I think there's two main areas. One is logic, sort of underpinnings of computer science. And there's some stuff here that's really interesting. One of my favorite theorems of logic is the compactness theorem, which says something along the lines of if you have a set of statements and you can satisfy every finite subset, then the infinite subset of all these or the whole set of the statements must also be true, which sounds quite intuitive. But actually turns out is occasionally not true. So if you look at, let's say, the set of statements I'm thinking of a natural number, but the number is not one. It is not two. It is not three. It is not four, etc., all going to infinity. Any finite subset of that set of sentences can be satisfied, because you're always missing either a number or you're missing the sort of the original statement of thinking of a number and it's not one of these. And so you can sort of satisfy every finite subset of these sentences, but it's actually impossible to satisfy the whole thing, because it is impossible for you to be thinking of a number that is not one, that is not two, and then not three, not four, not five, etc.

So I think logic is quite interesting, and that's one major intersection. Another one is sort of what are the limitations of computers and sort of what separates computer from a human. And so if you sort of go back 20, 30, 50 years ago, people would say, "Hey, if a computer can play chess, certainly they would be as smart as a human then." And of course computers can play chess much better than humans can. And then some people would say, "Hey, if they can play Go. I would consider a computer to be human." And of course they can beat us on Go now as well, right? And so thinking about sort of what the differences between humans and computers are is also quite interesting, but that's maybe a little bit further out. I don't think we're going to be getting sort of autonomous AIs anytime soon.

[00:43:24] JM: Cool. David, thanks for calling the show. It's been a real pleasure talking to you.

[00:43:28] DH: It's been great having talking to you too. So thanks.

[END]