EPISODE 1125

[INTRODUCTION]

[00:00:00] JM: Jupyter notebooks are the place where data scientists tinker with their models trying to make sense of large datasets. As these models get increasingly refined, they become ready for production use cases, such as recommendations, analytics and monitoring applications. And all of these are in use at Netflix. Getting a model seamlessly into production has many hidden challenges, which Netflix discovered in their efforts to build tooling around production rising notebook-based models. And to make this work, Netflix developed lots of in-house tooling, much of which was written by Matt Seal. Matt Seale has since left Netflix to start Notable, a company built to support the challenges and the creation and productionization of Jupyter Notebooks.

Matt returns the show to visit the challenges of working with notebooks at Netflix and to talk about the tooling that he is working on at Noteable.

Before we get started, I want to mention we're looking for a writer, or a researcher to help Software Engineering Daily produce articles and to create preparatory content for these episodes. You can send me an email, jeff@softwareengineeringdaily.com. I'm also looking for investments. If you are a company or if you have a company that is looking for investing capital for developer tooling or infrastructure products, send me an email, jeff@softwareengineeringdaily.com. And also, full disclosure, I'm an investor in Noteable.

[SPONSOR MESSAGE]

[00:01:33] JM: Open source tooling is generally preferable to closed source tooling, because with an open source tool, you're going to know what code is running. You're going to know what the community is saying about that code. And you're going to have flexibility. But scaling open source tools is not that easy. You're going to have to spend a lot of time managing and maintaining that open source software. And the alternative is to use closed-source software, which will be scalable, but you won't know exactly what code is running. And you won't have an easy migration path.

Logz.io is a scalable and fully-managed observability platform for monitoring, troubleshooting and security, and it's all based on open source tools like the ELK Stack and Grafana. Logz.io is open source software at the scale that you probably will need if you are a growing company. Sign up for logz.io today by going to logs.io/sedaily and you can get a free t-shirt. You can go to L-O-G-Z.io/sedaily, create an account for logz.io and make a dashboard. Once you make that dashboard, you will get a free logz.io T-shirt.

Thanks for listening to Software Engineering Daily, and I hope you check out logz.io. That's L-O-G-Z.io/sedaily.

[INTERVIEW]

[00:03:11] JM: Matt Seal, welcome back to the show.

[00:03:13] MS: Thank you. I'm glad to be back.

[00:03:15] JM: A while ago you came on to discuss notebook usage at Netflix. Could you refresh us? How are notebooks used at Netflix?

[00:03:23] MS: Yeah. Notebooks actually are used for all social things at Netflix now. But a big chunk of the use, they're used for exploring different problems and for building up these like integrations of different ETL tasks and operational tasks. So they use them very extensively for all the scheduled works. So scheduled data processing ends up running on top of a notebook as well as a lot of data visualization exploration tasks and a little bit of machine learning here and there.

[00:03:51] JM: And can you say more about the problems in infrastructure that surrounded notebook usage?

[00:03:58] MS: Yeah. Notebooks originally were built for the kind of code exploration problem where you might have very expensive operations and then some like repeated iterations against the results of that. And it sort of enhanced the idea of code REPL, where you have a live session where you're working code that has this mix expense. So the real use case first was for data scientists and being able to hold large amounts of data in one step and then iterate on a model or an idea or visualization against that. And so a lot of the early tooling in notebooks was really focused around that user group and that use case. But it turns out that the format was highly amendable for doing other integration tasks like data processing and expiration outside of the data science and machine learning-specific fields.

So a lot of the surrounding tooling for productionizing notebooks was not in a mature state at that point at the early point of notebook adaption. So a lot of people shied away from it, because things like it was difficult to port the code or make the code reliably re-execute. It was difficult to test. And they had this kind of file that we copied around a lot of places. And so there were some immutability issues. And a lot of those were types of problems that we were tackling at Netflix early in the notebook days.

[00:05:20] JM: There is so much data at Netflix. Is volume of data a problem at all in in terms of using a notebook or is that less of an issue?

[00:05:32] MS: It's usually less of an issue, because whenever you're doing really large data operations, sometimes you will pull them into the process. But it's no different than if you had a script that was running, like a generic I wrote a Python script or a SQL statement. And you are pulling that data into a single box. The overhead of the notebook doesn't really make that worse. It's no different than having a generic execution process.

So with big data, oftentimes what you do is you use things like Spark, or you use some data warehouse tool to aggregate data. So maybe you have like Druid or something else that kind of generates a view into the data that's useful for you. And you don't usually copy the whole data to your local machine. Occasionally, for machine learning, you do need to just deploy with really big machines. But even then, you kind of have to be careful about what you pull down, because you can pull more than it can fit on one machine, and it's sort of an independent problem from notebooks.

Most of the time you’re using notebook, you actually have Spark context and you’re running some query someplace or some other SQL interface. And so the data is living outside of your notebook, but you're interacting with it from the notebook.

[00:06:37] JM: The data roles at Netflix, how did the notebook usage get shared. You become a shared responsibility. So you have like the data analyst whose maybe developing the notebook or machine learning scientist, whatever you want to say. And then you have these infrastructure people that have to actually deploy the notebook to production. Remind us of the burdens that dichotomy in workflow creates.

[00:07:05] MS: Yeah. If you look at how they a notebook might be developed, it's a very similar vein to have other machine learning patterns might be developed. You might start something or you're not sure what the solutions are going to look like at the beginning. So you’re going to do an exploration phase. In this actually a common thing basically all software engineering goes through. It’s just that machine learning has a longer exploration phase that you come back to more often, because it's more ambiguous what you need to do to achieve your desired goal.

So when you start with a probably, you start exploring what the different tools are and what the data looks like, and you'll start doing things like sampling data, or running through some function that you want to see what the result is. And you kind of play with that until you get something where you're getting something close to the end result you want.

In the case with notebooks, you kind of do that exploration a lot. And at the end, you would have sort of a document that had all your exploration in one place kind of captured. But you still need to go through the productionization step. And other engineering disciplines also have this, but usually they have less exploration codes sitting there beforehand as I said.

So one of productionization challenges is really transitioning your development notebook code from a single, like large document that’s explored everything, something that reproducibly reruns. So it's really important you have something that can run end-to-end that’s going to be reliably executing the intended outcome. So you need to do things like cleanup code that you no longer need. Make sure it can rerun from a fresh state with no like prior environment context. Or things like maybe you ran something and you deleted it, but it was still in memory and you were relying on it, things like that.

In terms of improving that aspect, like we did a lot of investment in papermill and integration with schedulers and read-only interfaces and kind of transportation of these notebook documents, and that really helped in that type of problem space because in the productionization of your notebook, you could systematically prove that your notebook reran from beginning to end without the inputs or with real inputs. So you got the ability to kind of integration test the end-to-end result and kind of prove everything was okay before someone would take that and bring it to production. One thing that's a little different at Netflix though with that aspect is that not everything is handed off from one team to another. Many teams manage even though they are maybe a systems engineering team, they will still manage the full lifecycle of their experiment or their exploration. So they'll move it to production themselves. And so the idea was to make tooling to make that as easy as possible so people could self-service into productionization of notebooks and other artifacts.

[00:09:40] JM: Eventually, you spun out of Netflix and started a company with Michelle, who I talked to recently. Tell me about your motivation for starting a company.

[00:09:52] MS: Yes. So it was kind of crazy thing to do in the middle of a pandemic. I definitely second-guessed myself a few times about taking that step. But a few months later, I ultimately think it was a it was a great decision. The reasoning behind it was that large companies all across the tech sector, we're really having struggles kind of orienting around how to grow the notebook needs beyond kind of what has been described before. So a lot of companies kind of started on this productionization path. They adapted some things we published at Netflix what we did. They kind of built their own internal tools. And some contributed some of the open source. And there was kind of this splattering of solutions. But all of them ultimately were having difficulty staying focused. So the teams that were doing really good work were either getting changing direction or different business needs who are coming in that were disrupting their ability to kind of continue executing that plan to make their notebook experience better. And they also had a lot of struggle with shifting as their internal teams grew from just data science or their initial use case to extended use cases, extended users.

Not just Netflix, but many of the companies that are out there that were doing notebook work really struggled to adapt the multiuser aspect of notebooks, which is something that kind of spawned this idea of making a company that started from the beginning with the knowledge that you have a range of data users that all have needs in notebooks and you need to design from pretty early in the stage to build on top of the notebook experience to enable all those users at once in the same platform.

[00:11:26] JM: The platform that you've built, it includes notebook workflows as well as a data discovery system or a data catalog system. Talk about the tooling that you've built in your company, Noteable, in a little more detail.

[00:11:43] MS: Yeah. And we're still pretty early phase. So a lot of things we've built are in prototype and we're working with some early design partners to kind of hash-out the usability and making sure that it's a really stellar, awesome experience as we kind of grow out how many people are exposed to it. But I think maybe I'll talk a little more about what the problem is that we're solving and then the tooling kind of fall that from that of what you want to be doing.

So part of the problem that's trying to be solved is if you build a notebook experience for the sake of making a good notebook experience, you can make a really great tool for an individual. But you might not make a really great tool for team or an organization of teams to reuse. So, from that aspect, we really wanted to emphasize like real production and real organization workflows that you need to get through, which means you need access your data. You need to have that in the same place that you would do other actions.

One other real big appeals of notebooks is that you can do a whole bunch of things without leaving the page, without leaving the document, and you can share the result with someone else that can be rerun. And so we really want to lean into that and make it very easy as sort of a problem space to be able to connect to the data that you need to use and make it very visible to the user that's operating against it. And that's kind of like an engineering philosophy we're taking to the whole approach of how to build on top of Jupyter notebooks to add those things that are difficult to write, do an open source, but really enable real-world problem solving.

[00:13:19] JM: Just to hop on the business tack for a little bit, there is a bunch of companies that have been built around Jupyter notebooks. Companies that are raising money right now. Companies that have raised money already. And I'm sure there're plenty of incumbent companies that have built support and tooling for notebooks. How does the Noteable approach differentiated from the competition?

[00:13:44] MS: Yeah, and I think where we're really taking a stab at trying to do something that's a little bit different, and the reason why we decided to found this company instead of kind of just stepping away from all of it is that we really want to focus on the range of data users exploring outside of data science and machine learning. Making sure that users that are even less familiar with coding in general have a familiar place they could land to do lightweight coding that inter-opted with their dashboarding or their types of problems they're solving. So we really want to make a tool that's well-integrated into the other tools in your ecosystem and can play nice with your existing ecosystem of tools that you're familiar and gradually make it easier and easier to just stay within one operating place to do all your work.

The other thing we really focused on that differentiates us is a lot of notebook companies focus on data science, or a lot of companies focus on one user out of the many data users. So we're trying to take a step back and think more how do you assign some of these first principles in a way that really helps all the data users in your organization and growing to encompass those overtime to make sure that you don't have to have 10 different tools for your different data users to feel comfortable.

And I think that something that's been a differentiator from a lot of prior notebook efforts, or prior notebook efforts have had to try and change their direction to accommodate other users, or they really want to focus on one user, do it really well for that one user.

[SPONSOR MESSAGE]

[00:15:13] JM: Today's show is sponsored by StrongDM. Managing your remote team as they work from home can be difficult. You might be managing a gazillion SSH keys and database passwords and Kubernetes certs. So meet StrongDM. Manage and audit access to servers, databases and Kubernetes clusters no matter where your employees are. With StrongDM, you can easily extend your identity provider to manage infrastructure access. Automate onboarding, off-boarding and moving people within roles. These are annoying problems. You can grant temporary access that automatically expires to your on-call teams. Admins get full auditability into anything anyone does. When they connect, what queries they run, what commands are typed? It's full visibility into everything. For SSH and RDP and Kubernetes, that means video replays. For databases, it's a single unified query log across all database management systems. StrongDM is used by companies like Hurst, Peloton, Betterment, Greenhouse and SoFi to manage access. It's more control and less hassle. StrongDM allows you to manage and audit remote access to infrastructure. Start your free 14-day trial today at strongdm.com/sedaily. That's strongdm.com/sedaily to start your free 14-day trial.

[INTERVIEW CONTINUED]

[00:16:42] JM: The usage that you've seen thus far from companies outside of Netflix, how does it differ from what you've seen at Netflix? I mean, Netflix is one these very sophisticated organizations so that the tooling that can be built within Netflix is somewhat domain-specific by default. How does what you've built externally compared to what you had internally at Netflix?

[00:17:06] MS: Yeah. The funny thing is when we were at Netflix and kind of doing this promotion of how we're using notebooks and kind of when we last talked on the show here. Shortly thereafter, there is a lot of interest from other companies wanting to talk with us about things. And we sort of got together with different groups and try to get – Either meet them one-on-one and talk about what they were doing in the notebooks. Or we also had some sessions where we hold in groups of people from different companies that were all within the notebooks space trying to figure out what to do.

It turns out pretty much everyone have the same set of problems with the same set of domain issues and the same organization of data issues and user issues that they were all solving independently in mostly the same way. So it turns out that while many people think that Netflix is this monolith of perfect engineering and it has everything there, the reality is, is we were solving the problems we've just talking about a little more openly about what we were solving. But everyone in the space was kind of tackling these same core principle issues. And different people solved different core principle issues within that space to differing degrees.

I mean, I would say that for backend side of things, the integration with systems, Netflix did quite well on that front. But then other companies out there, like Amazon and Microsoft, did probably more on the frontend experience side of things with integration to their own stack, which was more relevant to other people. And the reality is, is that Netflix kind of stopped at the point of things that it needed for itself. And it was very clear there was a scope of problems beyond the things that Netflix was solving immediately or have the resources for that it could be applicable to everyone.

[00:18:48] JM: The sophistication of the data world, the data market, has increased pretty rapidly over the last five years, and yet it seems like the same problems exist today that existed five years ago when I started doing this podcast. Like you still have this problem of handoff, infrastructure handoff, and like kind of CI/CD workflow of data systems, like data models, data engineering. There's not as much of – CI/CD pipelines for some random microservice these days feels very smooth and ironed-out, and there's not really like many problems there. It's gotten to the point where it's like, “Okay, can you build a better UX for continuous integration? Continuous delivery?”

But in the data workflows, there’s still like acute real problems. We’re not at the point where you can just start like painting over these existing solutions with some glossy coloring and glossy UX. We still don't really know how to get past the problem of like, “Okay. I dump data into a data lake, and then I'm going to explore in the data Lake. And then maybe I’m moving into like a Kafka things or I’m moving it into Spark cluster and doing some stuff there.” And it’s like really convoluted. At least that's my sense. Do you do share that sense of the data landscape?

[00:20:09] MS: I think we are still pretty far from highly mature systems you can pull out of the box. I think that the really successful companies that have good data orgs do a lot of work that in 10 years, maybe 20 if the trend is slow, will look back on and be like, “Oh man! Why didn’t we have tool X and Y? This is so much easier now.” And I think it's a natural progression thing. I think data is more complicated than people give it credit for. It seems simple at first, but the combinatorial space of problems is large and a lot of the initial assumptions and standard engineering, other engineering disciplines, get violated and its data grows. So you get this always talking about scale, scale, scale.

And I think as a result, it feels like things aren't progressing. I do think there has been dimensions of the problem. They have gotten a lot more standardized. They're a lot better. They're not where they need to be. But, for example, I think that a lot more ETL has gotten standardized around scheduled work that has integration testing against it. I think there's a much more common pattern than it was a couple years ago. The scheduler space still is pretty contested for what's going to be a really good solution for everyone as they grow past certain data scale boundaries. But I think that some of the data tooling has gone a little more recognized, have as many new SQL engines for large data coming up as they're used to be. That there are still new ones cropping up trying to solve the problems a little bit better.

I would say for the part where we don't have really good – Like it's pretty much where it was several years ago, is for very small installations and setups. It's a little easier than it was to get up like a compute cluster and get your initial set up. But your data lifecycle and ecosystem around the compute is still very hard as a new company to get that established in a mature manner. You usually have a lot of resources or time or hire very specifically high-skilled people to put all the pieces of the puzzle together the make a coherent story for your company. And I do think that is probably further behind.

We're hoping some with notebooks will help on one dimension of that around if you make it really easy for the people who are less technical to interact with that warehouse, you do reduce one of the friction points that's there, which is once you get your basic ecosystem of and have a data interaction with your warehouse, usually from there, then you have to go build custom tools for every individual user group you have. And that's another big hurdle for people. And that's something I feel notebooks has an opportunity to solve. It's not the only way to solve it, for sure, but it is an aspect of that that I think will keep getting better with different tools in the near future.

[00:22:46] JM: You mention the scheduler layer. Are we talking about the data schedulers like Airflow, or Prefect, or Dagster?

[00:22:54] MS: Yeah, exactly. I think the scheduling on top of those systems for doing your data operations, like a lot of people have very ad hoc setups before and there's a little more consistency to that and there's, as you mentioned, some of the kind of common players people look at the start their with their scheduling systems.

[00:23:13] JM: Nick is pretty coy about Dagster, and I'm not sure how much usage it has. Have you tried that out yet? Have you tried out elemental Dagster?

[00:23:21] MS: I've been a fan of Nick's work with Dagster since like the day they started. I actually connect with them very early, and I thought the tool he's building had a lot of the right views into data problems mixed with scheduling. And so I've always been very bullish and excited by what he's doing and that front. And they've been kind of working through some of the data problems that they deal with, that they run into their customers and making a better experience over time. I think there's still a lot of open room for data scheduling systems to be made better and easier for different use cases, but I think the corner the Daster is trying to target is being well-addressed by their roadmap, or should be if they keep executing well.

[00:24:04] JM: So, you talked a little about the company's outside of Netflix and where they're at and the problems they have. I would like to know little more about what exactly are the systems and the services that are included in Noteable. What are the solutions that you're giving to people? Because Jupyter notebooks are open sourced, and what are you kind of adding on top of that open-source notebook layer that adds value?

[00:24:34] MS: If you actually look and you talk to like – We've talked to dozens of companies and hundreds of people on like how they use notebooks across the past few years, and especially since we started founding Noteable. There's this really common consensus of really fundamental and very specific problems that come up with trying to host your own Jupyter. I think having an open source Jupyter hub that you can bring up and it has Jupyter Lab and extensions in the kind of ease of getting that up has been made really – Like has been really well done by like specially some of the folks over at Berkeley and make that like super simple.

So I think it's a really good starting place for getting an ecosystem up and running. And for some people, that's sufficient. But you very quickly run in all sorts of integration issues where you the open source kind of doesn't have a specific answer for that integration. You don't have direct access to your data system from your notebooks. So you often have to try a lot of boilerplate code to connect to your data. And you need to do that securely. You need to do things like quality of life for extended things that work in Jupyter, but don't have a good open source extension for displaying them. All the way down to like little things like what's your experience with how SQL looks in your notebook, right? Normally, you don't get nice linting, or auto-complete, or visibility into the data you're operating on. And those things, you can kind of slowly add to open source. But what we're really aiming at doing is connecting all of your production workflow patterns through the notebook.

And I think in terms of solving problems, we want to make sure that the fundamental things like how your notebooks are versioned, how they're integrated with your source control? How you can easily integrate with the scheduler? How can you easily run all of the different patterns you want without having the UI get the in way, where in some places the UI was designed for one use case? So kind of want to keep the protocol, but we want to really extend in these common patterns everyone hits around data governance, notebook governance.

I think the other thing that really comes into play is when you go past having hundreds of notebooks, it becomes very hard to track what's going on. So a lot of enterprise systems also focus on traceability and awareness of what's happening in your ecosystem. That's something that isn't in the open source, because for most cases, it's outside the scope of an open source solution. So those are the types of problems we're trying to solve. And we have some pretty good steps to them already and we've got some amazing prototypes, and many of the things are already implemented. But we're still kind of working through with [inaudible 00:27:06] really say exactly how they're going to work in the open after we get feedback from our early partners on how well it goes.

[00:27:15] JM: The market is, as I said, fraught with a lot of different companies that are attacking this. Do you feel like what you've built thus far with Noteable has a good enough shot at being dominant? Because it's like I feel like a lot of these different solutions could sort of bolt-on the tooling that you've built with Noteable. And I just wonder how you're going to navigate the competitive dynamics of the business of making a notebook platform.

[00:27:51] MS: Yes. So, I actually see that in the ecosystem of all the notebook offerings, there's actually yellow a lot of different companies that focus on niches of the problem, of problems for different data users. And there's also targeted for solving for like the general IC case of someone who wants a better user experience within just a single notebook.

I think where Noteable really bring something new to the field is that we know how these things need to integrate in enterprise systems, and we know how to build tools such that you make it very natural for the whole ecosystem and organization to inter-op with a notebook, which means your data experience within the notebook should be unparalleled compared to the offerings. But we won't have lock-in to our offering. Since we're building everything on top of Jupyter and we're making it all compliant with Jupyter specs and being able to ground-in other Jupyter contexts, it's one of those things where if someone wanted to use a different tool, we would play nicely with it. And we don't want to be one of those all or nothing tools where somebody could come in and only use our tool. I think that's been a death of a lot of companies in this space in the past as they try to reinvent the entire world at once. So you had to do everything their way or nothing, and that really makes it hard for early adaption and it really makes it hard for transitions to using different tools within the ecosystem or adjacent tools. I think we have a good knowledge of all those interactions and really good knowledge of what's needed in a user experience to make that data usage of real-world applicable.

[00:29:25] JM: the scheduling problem that you solved with papermill at Netflix, is that part of what you're offering with Noteable? Or are you focused more on the data catalog and data usage side of things?

[00:29:38] MS: Yeah. I think, obviously, we're going to know how to integrate and make it run you're your scheduler. I think that's kind of a given. We're not targeting beginning a scheduling company. So the intent is to make it easy to use a scheduling system you're familiar with.

All right. So on the scheduling case, we're definitely going to bring the knowledge we have to integrate papermill and make that a first-class operation within Noteable. I think one of the things we're also going to bring is the awareness of how to play nicely with lots of different scheduler systems and not force a person to kind of abandon what they already have or reinvent the wheel that they've already designed for their own system. So we're going to emphasize a lot on being able to play nice with the existing schedule that someone has and making it very smooth to use.

And I think actually that's been a really powerful aspect of the work we did with papermill and the schedule integration talks that we did, because the work we did – It was really easy to use papermill in almost any scheduler. Matter of fact, even things like Airflow publish the papermill operator just in their open source, “Here's how you use it.” And that was actually a thing that really triggered a lot of popularity with that tool chain, is that the ability to use it anywhere was very high. So, all sorts of systems could adapt it with ease.

What we’ll probably do is make sure that you have a tighter integration between a scheduling with your notebook system so that you don't have to leave your system to kind of know what's happened in scheduler and be able interact between the two much better.

[00:31:08] JM: As you’re engaging with these companies that are potential customers of Noteable, what have you learned? What have you learned about how their infrastructure is set up and how they are using notebook? I mean, whenever I ask this kind of question to somebody, the answer tends to be, “Well, we’re surprised every single time we see some deployment there always using something in a weird way.” So what are you seeing?

[00:31:35] MS: Yeah. I mean, so far, a lot of it has been pretty standard. A really common thing that’s been asked a lot is people just want something that's a little bit better than the open source. Like all these fancy things we’re going to do and these like awesome integrations and things. Honestly, a lot of people are at the stage of their employment and integration lifecycle where they need like the most basic interaction improvements. And I was even surprised on things with like how much people cared about the visibility and that the governance of like what even are these notebook? I mean, it’s not surprising, but it is in its own way. Just having the knowledge of where all the notebooks are? Who owns them and how often are they used?

The most basic [inaudible 00:32:20] are something that's like one of the most commonly asked for things that we kind of just assumed was for granted. Doesn't everyone build this or have this? And I think it's something that was a little surprising like that alone gets people interested in using a new product. And I think that's a really low barrier.

So the barrier being so low is lower than we expected. I think it gives credence to why launching this company that solves a lot of these basic problems and then start solving the really hard problems had a lot of value add for us, is that we know those problems were going to be solved on day one.

[00:32:56] JM: Tell me about the necessity of sharing notebooks. Like one company having a notebook and that might be used as a template for a lot of different things, and other people needing to use that template. I'd love to get into some usage of notebooks that people – Usage and workflows that different companies are using it for.

[00:33:17] MS: Yeah, and there are a few interesting ones. I believe you talked some about the Netflix, and then I'll talk to them about some other use cases that Netflix didn't use as much. But in the case where you kind of promote a notebook to a template, you can think of this as elevating the level that you're sharing the notebook from like individuals you're sharing with to a whole org and giving it some first-class attributes, like making your scheduler know about that template by name and know all the arguments it needs or the parameters it needs to execute.

And in this promotion process, you can sort of get these like golden kind of stamped versions of notebooks that are available in immutable form that have a well-defined characteristic of operating. So it's a complete integration operation task. And when you get some of those, you can – What we would do at Netflix is we exposed these templates and we let any team contribute. So a lot of teams contributed their own templates for common workflows across several teams they supported. And then all they have to do to tell a user was they just submit this notebook. They kind of go through a review process. They write a test for that and they promote it through after they kind of done all the things like documentation and things in not notebook that we pull out.

And then from there, all of a sudden, all their users could just specify, "Oh, here's the name. It has all the arguments you need and just need to fill them in with – Here's a clear auto-generated dock, and everything is there for you out-of-the-box. And that was a really easy way for people introducing new common patterns. So something like a really – The most basic one is my Spark template, where I want to run a Spark job and then I just want that to be able to pass in a simple

Spark query and it just runs. [inaudible 00:34:57] the option to configure anything. That's a really easy thing to build this template for and it’s very visible what you're doing.

Then some more complicated things you might do might be like some sort of synchronization from like a common source. Like maybe you have a vendor who's generating – I’m making this up, but a vendor who’s generating CSV content and you need to transform that. Do some filtering, or maybe just even move that from that system to one of your other systems where you have a more mature pipeline to run after. And those little like get data from A to B is a super common pattern that usually isn't that much code. Sometimes it is more if it's a system that doesn’t have a good library behind it. But usually, you just have dozens of these little use cases where I need to get data from system A to system B, and that’s a perfect use case for a notebook template.

In terms of sharing, I think one of the really important attributes that enables that is you need immutability and you need read-only modes. So you need to have everything you run referenced from the immutable source. What I mean by that is that your version that you’re running for your template or your notebook is not going to change between executions. You might, in the next execution, choose the latest version, but you can always run version 647, and it will run that exact code and you know what that code is. And that’s a really important attribute for productionization and maturity of scheduled integrated system.

And then read-only is also super important. You want to be able to read the results or read the template without knowing no one can edit the source or truth, or edit what the results were. If you want to get new results or you want to change it, you always make some copy or make the new version. And from there, they can kind of play with it as they want. But it doesn't affect the original source of truth. And those two attributes really enabled those kind of well-productionized, shared, template notebooks, and they’re pretty simply. You just need to make sure you put the rules and the interfaces and to enable that.

Outside of Netflix, there were lots of interesting use cases, things like – There are some great ones around security exploration. People would do system inspection and kind of run through a checklist of things to look for intrusion or anomalies within their system either for security purposes, or even as a data analysis thing. Finding anomalies of data that they’re being given or they’re reproducing. And that was really interesting use case where you kind of run that periodically and kind of have an alert trigger when something is out of band, or as a template for a runbook.

We did this a lot actually at Netflix. I know other people did in other places where you can actually make all your kind of PagerDuty runbooks, just will notebooks you can run through. You can document manual steps you need to take. You can put automated, like do these API calls with this input here. And you can kind of capture what someone needs to run through in an emergency situation to kind of repair system that's in dire straits. And those are like a couple really common examples of productionization of notebooks that could be done well.

Oh, I forgot one aspect that you really want with all of these. You want to be able to write an integration test. So you want at least one test, preferably a couple, that runs through the logic of the notebook end-to-end. For that, you need to keep the notebook fairly linear. Otherwise, your integration test will have a hard time covering the branching pattern and the edge cases that you can run to within the notebook. Outside the notebook, the ecosystem you’re interacting with might have lots of edge cases. But you want to prove that the notebook logic is sound, and that when you run it it’s going to reproduce clearly run.

And a really common thing there to do is just schedule a papermill job to run your notebook that you're going to use in the real setting with some dummy inputs and just run that every time. Anything related to it changes, or on some schedule. And then you have a reliable way of knowing that your notebook is going to be okay.

What's different from what we built our systems that's now available as well is unit testing and notebooks is more capable. So there is a library. I helped a Google Summer code student work on called Test Book, which is in like kind of the first release cycle of features. And that enables doing unit testing into [inaudible 00:38:58].

Today, that’s only primarily supporting the Python use case, but it's very easy to extend it to other languages. So now you could have more complicated, like almost library functions within there and you could test and ensure they work. And I think there’s going to be more tooling like that to make interoperability with notebooks and standard code practices easier.

[SPONSOR MESSAGE]

[00:39:26] JM: Operations teams can find themselves choosing between two options, either take full control over the infrastructure yourself or give all developers permission to access production. The first approach increases the operations teams workload, which often results in overwhelming situations and ops becoming a bottleneck. And the second approach allows for rapid deployment of changes to production, but it causes a serious risk to infrastructure uptime.

With Octopus Deploy, operations teams have a third option. The Octopus platform can be authorized to run approved steps and play an intermediary role. Octopus delivers self-service without sacrificing control over production, and it also provides a comprehensive audit log of the changes that are being made. Developers can enable self-service with automation. By automating the processes that are forming a bottleneck, developers can free themselves from the waiting game.

You can check out the most frequently requested runbook templates by going to octopus.com/selfservice. You can find those runbook templates at octopus.com/selfservice. And I want to thank Octopus for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:40:41] JM: The developments in the machine learning space, this space is obviously moving really fast. Are there any opportunities to leverage all the changes going on in the machine learning ecosystem as a product designer for Noteable? What do you see as an opportunity businesswise in making better machine learning experiences through your notebook platform?

[00:41:05] MS: For machine learning, there is a ton of great new libraries that come up. And I think it's been a rapidly evolving space. But more and more, libraries are coming out of the box with visualization-enabled by default. And a really natural place to do that visualization is within the notebook, because you don't have to have some export you're going to look at or some other system where you're going to dump the results you need to analyze.

So more and more, machine learning libraries are actually coming out of the box with support for outputs for Jupyter that render naturally. And then you get a much richer evaluation experimentation pattern where you need much fewer lines of code in order to a get rich representation of the state of the problem you’re looking at or the state of the result that you're looking at.

And for the tools that are out there that are up-and-coming or already established, I think that's like a really winning aspect of notebook integration that if you're looking at solving for that corner of the data user experience, then making sure these tools behave really naturally within a notebook experiences is critical. You might do things like make sure if they do have visualizations, that they get captured cleanly and rendered nicely. You need to do things like if they don't have a visualizations but they have the data for building a visualization, maybe automating when that is run in the notebook experience to additionally trigger off visualization rendering for the user and making it so that the user can do visualization with less code. I think that's an aspect that's proven very valuable for lots of people. And that might even be in the surprising fact side of things that auto-viz or even just the easy-viz is super valuable to people, because you waste so much time trying to fiddle with getting the X-axis right or not quite selecting your data correctly and having a UI that just renders automatically to show that I think makes those tools much more usable. And notebooks are unique and that they have that visualization built right next your code execution so you don't have to leave the tool to do that.

[00:43:08] JM: The productionization of notebooks, I'd like to revisit this. What are the difficult parts of getting a notebook into production and maintaining that notebook in production, updating it over time?

[00:43:25] MS: For notebooks in general, I actually reverse it from the other way around. I think if you start with a lot of people get stuck on, “Hey, how would I productionize a notebook relative to this really sophisticated library with large amounts of DevOps tooling and CI/CD tooling already in place, or lots of best practices.

I think, actually where the problem is usually we have a lot of those tools in place. Productionizing a notebook is like productionizing anything else. You make sure you have a test.

You make sure you have code review and you have a deployment cycle that controls when that notebook goes to users. And beyond that point, the and the standard execution environment is pretty normal. In this case, you're usually doing integration tests and you're usually doing some sort of [inaudible 00:44:07] based promotion to whatever your production line is for exposing that asset to a user either in automated or a manual fashion.

And when you actually look at it from the reverse side, look at actually how most orgs have data usage. And you were talking about like things are still kind of not great in data usage tooling. A lot of orgs, even very big, sophisticated orgs have a million S3 files. They have some Python code in them that do their ETL, and there some place and they're copied between places, or they're in someone's hard drive or an on-prem system. And they just have these scripts that are everywhere.

And if you come at it from that point, those users have been neglected from tooling. Tooling is, honestly, kind of hard to get set up, and it's 1 to 3 context shifts away from the problem the these other users, like data analyst data engineers are focused on. They're not focused on best practices around programming tooling. They're focused on getting a data results to the business to analyze. So they care much more about like dimensions of the data they're working with and like the consistency of it. And then tertiary, you can start into things like code quality or version control, and they're gradually growing into using those tools and there are things to make them smoother so they don't have to think actively about using those aspects.

If you start from that POV of a user, you had a strict someplace that stash – I don't know where. And you run that for your production, and then that person moves teams or leaves the organization. And then your ETL pipeline stops working one day. You have variable visibility. So your productionization of that script was pretty poor.

If you think of how a notebook, a notebook could be used in the same way, where I did some develop in my notebook, in my personal space. I ran a schedule with it. And then I kind of walked away. Or I run every morning on Monday. I manually go run my notebook. This is a more common pattern than people care to admit. I think when you want productionization, you really want to make it easy to get away from those patterns without the user having to think a lot about what they're doing.

So one thing making very visible, the versions, like automating the scripts they're doing to be into some sort of linear or actual version control system to get visibility into changes over time without the user having to change their workload substantially. Also, making the tooling for how you're interfacing

One thing that made productionization of notebooks much more successful at Netflix was users could start using notebook templates without knowing they were using the notebook. They didn't have to know what a notebook was. It was a tertiary concern with them until they got used to using one. But what we did was the way you execute it, your result would be a notebook. And if you look at what ran as the source, it was a notebook. And you just had read-only copies of all these things. And what we put to people for how to debug, you want to productionize the lifecycle of something. So you need to make debugging easy. You need to make re-launching the environment [inaudible 00:47:03] easy and you need to make it easier to review and analyze what changes have happened.

So you can kind of make chip-away at these problems from different angles. But being able to see what ran, being able to reproduce the environment without things you're doing in the actual, just clicking one button, I should be able to rerun in a live environment what I ran in the scheduled environment. That was automatically built for me.

That's where you want to pull all the user's tertiary concerns into a common pattern that helps the productionization from a coding point of view or a code engineering point of view without encumbering the user with having to learn 10 systems. I think it's where a lot of people get tripped up, and I don't think it's specific to notebooks. I think it's just that people leaned into notebooks currently often out to users who were in the groups that were coding best practices were tertiary to their primary day job concerns.

[00:47:54] JM: The whole idea of generating new notebooks from immutable read-only notebooks, it reminds me of the – I think it was a post by Max [inaudible 00:48:07] a while ago about the immutable functional data engineering. Does that resonate with you? The idea that you have these immutable structures that you reuse and you generate new structures from those even with as high-abstraction as a Jupyter notebook?

[00:48:25] MS: Absolutely. If you look at the structure of a Jupyter notebook file, it's a well schema JSON document that you can think of as little tree structures. So it actually mirrors a lot of those types of problems. So you don't really use it at that scope for the type of problem we're talking about with immutability. But it is there and is corollary.

What you have with the notebook is usually a very small source document. It's not like this gigabytes of data type of problem, even though it might be processing gigabytes or terabytes of data. The notebook immutability mirrors that a lot. So I think you saw kind of a wave of immutable coding practices in different places. But actually in terms of system engineering, making immutable artifacts, so like this is what versioning and packaging was doing very early.

I load version 1.12.3 of my package. I know exactly it's in that package. I was shot and know what it is. I'm not going to accidently get some slight changes that shouldn't affect anything, but ultimately ruins my Friday with taking down my system. You have the same thing with a notebook document. You want to make these versions of the notebook that kind of follow through history of what was the state of the notebook when it ran at a certain point of time.

And if you think of that as either get version control, or linear versioning, you get a similar set of resulting attributes of the system that you get with immutable data with data systems. The difference being that immutable data systems have the extra problem of having to solve for the fact that the amount of data you have can start costing a lot of money. So, with immutable data structures, you have to be conscientious of how do you efficiently use space and how do you efficiently kind of clean up or a wrap up immutable actions in a way that is either encoded such that you don't use up a lot of extra space or you throw away old histories so that you spend a bunch of extra money.

The notebook side, you can have tens of thousands, hundreds of thousands of notebooks and usually your expense to that is like a .001% of your expense compared to the data side that they're offering against. So it's actually even a little easier in that constraint to think of.

[00:50:30] JM: As we begin to wind down, do you have any predictions for the future of notebook infrastructure or even just data engineering infrastructure more broadly?

[00:50:40] MS: I think that a lot of companies are going to be adapting more mature data lifecycles. In the past few years, I think it’s become very apparent to many kind of – You wouldn't call them tech companies, but companies that have tech infrastructure that their data quality of – Their ecosystem around their data is causing data quality issues. And as a result, there's been a lot more attention on like the pyramid of needs in the sense of your data system. And that there isn’t just if you pull out solution A and everything is solved for all of your data problems. But, really, solving fundamental needs, then intermediate needs, and then advanced needs. This would be mapping to something like in the very basics of, “I know where all my tables are and I know when they're updated.” That might be very minimal thing. Than you might have a layer above that of, “Okay. Well, I know when data lands that I have no nulls in my non-nullable call.” Some really basic things that I know are going to be there.

And then maybe a layer above that might be I know when data lands that is valid, but incorrect. So you have more checks for audits before you release downstream actions for users. And I've seen a trend of a lot of companies thinking and talking about this domain of their problems a lot more. And before, it was kind of isolates in maybe the top 20 or 30 tech company that really dug into that and tried to solve those problems. So I envision the tooling supporting that, and above that will be very critical the next few years. I think that’s a place where things will grow.

[00:52:15] JM: Matt, thanks so much for coming back on the show, and congrats on the company.

[00:52:19] MS: Awesome! Thank you. Thank you for having me.

[END OF INTERVIEW]

[00:52:29] JM: As your infrastructure grows, you have SSH host and Kubernetes clusters in staging, production and maybe even on edge locations and smart devices. You want to implement the best practices to access them all, and that can be time consuming. Security tools can get in the way of engineering work. Well, there is a tool called Teleport. It's open source software. It's written in Go. It's a drop-in replacement for open SSH. It also has native support for Kubernetes. It's built by Gravitational.

Teleport provides identity-aware access using short-lived certificates with SSO, session recording and other features, and it ensures compliance and audit requirements. Teleport also prioritizes developer and convenience, because it's built by engineers for engineers. You can give it a try going by going to try.gravitational.com/sed. There are links to downloads, documentations, and a GitHub repository. That's try.gravitational.com/sed to try out Teleport from Gravitational.

Thanks to Gravitational for being a sponsor. And again, if you want to try it out, you go to try.gravitational.com/sed.

[END]