# EPISODE 1098

[INTRODUCTION]

**[0:00:00.3] JM:** M3 is a scalable metrics database originally built to host Uber's rapidly-growing data storage from Prometheus. When Rob Skillington was at Uber, he helped design, implement and deploy M3. Since leaving Uber, he has co-founded a company around a hosted version of M3 called Chronosphere.

If you have access to a scalable metrics database, you might as well start accumulating as much data as possible, right? Well, not exactly. If your company generates enough data, you probably want to turn down the dials of how frequently you save those metrics and down-sampling will reduce the amount of money that you pay for these hosted metrics. You can choose how much to sample.

In today's show, Rob discusses the engineering and deployment of M3 and how that work led to him founding Chronosphere, as well as the product offering of the company. I do want to mention, I invested in Chronosphere. If you are building a software tool for developers, or some infrastructure product, I'd love to hear about it. I am looking for companies to invest in. You can e-mail me at jeff@softwareengineeringdaily.com.

I'm also looking for writers for Software Engineering Daily. You could send me an e-mail, jeff@softwareengineeringdaily.com if you're interested in writing for us. With that, let's get on to today's show.

[INTERVIEW]

**[00:01:24] JM:** Rob, welcome back to the show.

**[00:01:25] RS:** Hey, Jeff. Great to be back. Good to see you in these times.

**[00:01:29] JM:** Yes. Likewise. Not talking about anything pandemic-related, although I know you are – you've been sheltering at home for a while, building a company, building a company from home, so maybe we can get to that a little bit later on. The subject is M3 and what you're building with Chronosphere. M3 we've covered in some detail on previous episodes, which is I think the boilerplate way to put it is Prometheus scalability.

I would like to spend some time on talking about why Prometheus needs scalability. We've covered that in a lot of depth in the previous episode. Now I'm thinking maybe in this episode, we can talk more about how to build it as a cloud service and how to how to build the kinds of fault tolerance and scalability that you need to productize an open source project. Let's just start with just a discussion of M3 and rehash some of the monitoring issues that you encountered at Uber and how that turned into an open source product.

**[00:02:30] RS:** Yeah, most definitely. It sounds like a great set of things to talk about. Just to rehash M3 and where that started, as you said we have covered this topic before, but in a succinct summary, it really is an open source metric solution that was built out of a need of a growing set of monitoring data being generated from Uber's move from physical servers to containers. That happened over the course of four and a half years at Uber; the development of M3 and the migration to it from an existing graphite stack.

Also part of general Prometheus installations were also running in different small pockets of the company at the time as well. M3 consolidated all of those different monitoring solutions that the company had into one data plane. That data plane basically collected metrics in a central place for every piece of component that the business ran; your databases, your containers, but also a whole bunch of what we called business metrics. So things that were needed to be monitors that weren't technically performance metrics about the software, but how the API was performing the responses that have returned, the types of queries and transactions that were happening inside the Uber system itself and even tracking things, like how well the search algorithm was working in certain geo areas and things like that.

It was really used for a pretty much a wide range of things. Definitely helped on rolling out experimentations. You see, Prometheus is being used for all of this in the wider ecosystem. Also, because Prometheus works so well on Kubernetes and cloud native, that's why it's also a to fault getting a whole bunch of attention and popularity. M3 was really following that train and evolution internally at Uber. Then we open sourced it since we really did believe that by centralizing all this data in a long-term storage for Prometheus, that was of benefit to the wider ecosystem and we wanted to work on one system together with other companies, rather than reinvent our own wheel and keep it in proprietary land.

Then it took off from there and it's being the communities – community behind it is exceptional and yeah, I really love working on it and super glad to continue working on it, building a monitoring company now on top of M3.

**[00:05:16] JM:** Can you give a little bit of the landscape of alternatives and just how it compares to the Datadogs, or the new relics of the world in terms of usability, or scalability, or cost, or I guess, just the open source nature of it?

**[00:05:38] RS:** Yeah, most definitely. Comparative to solutions like Datadog and other large existing monitoring companies, M3 is purely focused on I guess, metrics. It has developing integrations with joining that metrics data or to trace data and things of that nature, which we gave a talk on at KubeCon in San Diego about deep linking metrics and traces.

However, yeah, it is really focused on metrics being at the center of your universe with your monitoring strategy. Whereas Datadog provides logging metrics and traces and you pick and choose which are important to you. The way that we think of monitoring your metrics is that it really is the glue that holds, provides the 10,000-foot view into your system. Logs is extremely valuable in debugging point in time events.

For correlations and for understanding at a high-level what's happening, metrics we feel is the best. There's a reason why alerts are done on top of metrics, it's because metrics has a high signal-to-noise ratio, because you're taking all these events that are happening in the system, you're picking the few that matter out to you and you are very carefully putting the right tag names and tag values on them. A human can understand the volumetric types of things that are happening in the system. Huge spike in latency, huge – tons more CPU being used, more network being used. Or if it's a certain category of pods all restarting around the same time.

Anyway, coming a little bit, circling a little bit background to that was a little bit why we believe metrics is so important. M3 itself is an open source project and focused on metrics. Datadog and other typical large monitoring companies, I guess, they're not really built on an open source solution that they have out there, that they’re working on and improving for the community. That's something we're thinking about.

Then there are other motoring companies out there with their own open source projects as well. However, we are really focused on providing the same level of scale, then similar to problems that Uber ran into. I guess, that is what differentiates M3 in comparative to other Prometheus long-term storages.

I think that a lot of people that just want a long-term storage and they don't care as much about the reliability, or the scalability, there are plenty of other options out there. Thanos is a great open source project, which stores a lot of Prometheus metrics in a file store, so something like S3, or GCS. That type of project is great for people who don't have a whole lot of like online requirements.

If you're building things that you want to publish your metrics back into your product, so you can imagine you've run some software for other people. Maybe you run a database as a service company and you want to show your customers the CPU utilization and the memory that database is using, or how many transactions they're doing, stuff like that, you want high reliability for that, so you want replication. You don't want to just be throwing these things into an object store and relying on that type of a back-end for serving those metrics directly back to your customers.

At least, that's the way we view it. At Uber, it was it was all about the fact that machines move around, computers restart and we never wanted to take a gap in availability for our end users, because we knew that 10 minutes of downtime in our system could mean hundreds of thousands of people out of work, that couldn't use Uber. That's where the high reliability underpinning goals really came from. That's why we always replicate data at three times and we replicated immediately. You can take an outage of any one of your single machines in the pipeline and the solution will keep going.

Yeah, it's structured very differently to your existing monitoring systems. We really are taking this thing of creating one single primitive that's very, very powerful, similar to how Confluent work on Apache Kafka and that is the underpinning of many data pipelines that people work on. We think of M3 as the underpinning of just doing monitoring and observability and alerting in a scalable, reliable fashion.

Chronosphere is about providing that to people at that next level of cost efficiency, reliability and scale, using that underlying primitive, but also making sure that that primitive is available for the community to build upon as they experiment with their cloud native and Prometheus deployments.

**[00:10:55] JM:** Since starting Chronosphere, has your work primarily been figuring out how to productize this thing? Or have you also had to develop additional work in the open source project, I should say?

**[00:11:10] RS:** It's been a little bit of both. I will say that productizing the work is definitely like what a large percentage of our time is spent on. Users don't really want to press a button, get an M3 install and hook up their Prometheus installation to their back-end and run Grafana themselves or anything like that. They want an all-in-one solution. If that works a 100% compatible with their existing Prometheus installation, that's great and that's why M3 is such a good starting point.

That's where they start off at and why M3 and Chronosphere is great is because it's a very natural transition. In fact, you run both systems in parallel, your existing Prometheus installation as well, as piping your data to Chronosphere. However, that is the managed offering. They don't want to continue to run a lot of different infrastructure themselves.

In fact, managed offering has a collector agent, which essentially is deployed as a daemon set in Kubernetes. If you're running Kubernetes and it basically takes over all the active monitoring aspects for you. We have our own visualization UI and data source UI. You can plug in Grafana and it's in fact, you can definitely use Grafana with Chronosphere hosting product, but also we allow for you to do things in a more managed way, which is user UI to configure which pods gets scraped. Use different strategies for keeping your metrics at different retentions and resolutions.

For instance, part of the power of M3 is that you can choose to take subsets of your metrics volumes and store them at different lengths of time, so you don't suddenly have to take the firehose and keep the firehose around for one year. If you want to look at metrics from a year ago, you can really opt in a whitelist, certain things that you care about enough to keep around for a year.

This managed platform is about making all that super simple without having to read a whole bunch of YAML files and have three SREs splunge around with your Kubernetes cluster and configure everything. The power of us starting with M3 is you can start with that type of installation that you've carefully curated with Prometheus and move that existing deployment over. Then piecemeal use some of the nice managed parts as much as you might like.

A lot of people, at least even start onboarding with us, just because they want to get out of the cost and the management overhead of their really large Prometheus installation that grew over time is too much. They love Prometheus, but they also would like to have a cost-effective reliable scalable back-end. That means they can just go starting with that, where the business is today. They can grow to the scale of Uber if they wanted to and then still be paying a linear increase in cost with related to how big their metric volume is.

**[00:14:21] JM:** Let's say you get 5,000 new customers in a single day. You just have some amazing day, where 5,000 new peep install the agent in a daemon set and start scraping a bunch of metrics and you very quickly have to scale up. Give me some picture for what your back-end looks like for how you are hosting all these M3 instances and how you have tested the scalability of that provisioning system.

**[00:14:53] RS:** This is a 100% what we spend our days doing, standing up the managed platform was all about making sure we had an isolated environment per tenant that can ensure that and that scales under the hood without you needing to any of the specifics. The nuts and bolts of that system and making that scale, providing the level of isolation required for all these customers with high-end needs has taken the core of our focus.

Part of that focus of course has bled into us needing to develop more and more M3 features, which go back into the open community. The other parts of it is much more along working out

how customers engage with the platform and how we react to their scaling needs. Right now we have a model where they can request how many provisions metrics per second they want to collect. Then they use a slider to set that number.
We allow for some amount of burst capability. But overall, the burst capability is limited to a certain time window. That way, we can provision exactly the right amount of resources underlying right capacity in 3B clusters for their workload. Basically, it's a mapping of all these provisioned metrics that people have, metrics per second have configured to the retention resolution they want to record it for, so for how long they want to keep it for and for how fast the sample frequency is. That underlying metrics per second per tenant all get added together and mapped to underlying resources.

Then as they adjust those sliders, we essentially work out whether we need to add an extra storage node. If we do, that essentially is what happens in the background. If a customer exceeds their rate limit, that's when we start to drop some of their metrics and they can choose different ways to do that.

They may say, “Well, I don't want to drop all my metrics. When I hit a rate limit, I want to specifically drop some metrics that I don't really care about.” Nearly as much, they're just nice to have. You can imagine extremely detailed, granular stats about a piece of the infrastructure. That's really nice to have. You may want to drop that, instead of just randomly drop bits of your metrics. You become very reliant on these metrics. You do not want an alert going off, just because you had some person at your company onboard a whole bunch more metrics, because you're dropping random data, or you do that activity, that change in your system.

We have a whole bunch of these rate limiters that essentially can perform things in a smart way to make sure that we're dropping things that aren't going to interrupt your day-to-day. Things like that is a large amount of our focus. Even just the routing. We have a global load balancer and that depending on the tenant, will route you to at least certain isolated gateways. They share underlying resources, but they are network isolated and process isolated, if you want a deployment that is truly isolated. We do have some extra powerful features for customers that want extra isolation. Then that's something that's super valuable to them that they can't find in other products.

**[00:18:36] JM:** Do you need to make M3 deployable to both GCP and Azure and, or I guess all three major cloud providers? Can you just obfuscate it? Can you just make it on AWS? Does it matter?
**[00:18:55] RS:** Yeah, great question. I mean, we are multi-cloud in nature, so this is something that day-in, day-out is a challenge that we solve. Right now, we're deployed pretty much primarily on Kubernetes. We do have some parts of our stack that are cloud provider aware, but the rest of cloud provider agnostic. This is a nice side effect of M3 being developed at Uber, where a large amount of the infrastructure was on-premise actually for a significant amount of time, is that we really are cloud agnostic at the core layers of our offering.

We would love actually to add to the open source project a way to archive unused metrics back into the cloud and slowly bring it back in, but that's never been a real priority for us. It's really about being able to do a week over week queries with everything available and super-fast response times. Especially when you're doing anomaly detection and stuff like that, you need that level of speed on historical data.

Anyway, back to the implementation on running on top of cloud providers, so yeah, we run on GCP, we also have stacks in AWS. Azure, we have – we've tested with. We've run a stack there, but we took currently don't have any customers that really want to production environment in Azure, but we welcome whoever. If that's something you want to try out, I would love to talk to you.

In terms of the complexities of running even just in GCP and AWS, it really comes down to for us the gateway and the load balancer that gets you to our Kubernetes cluster. Because AWS and GCP have very different load balancer options. For instance, GCP has a global HTTPS load balancer that can do SSL termination for you. That's pretty fantastic. We can actually rely on GCP's underlying infrastructure to provide general ingress for whether a collector runs in any part of the world and it showing up correctly and in the cluster that we run their stack in. That's something that with AWS is a little bit harder.

We use GRPC for all of our collector end-to-end publishing of metrics from your agent that runs wherever you are, so you could be in Azure, or AWS, or GCP and you run your collectors there. They use GRPC to push metrics data to our back-end. On AWS, there's an HTTPS load balancer, but it's on the compatible with HTP1. I think it's something about the HTTP. It does support HTTP2, but trailers are broken and that's the trailing headers that are returned.

The problem with that for us is that we – GRPC uses that feature for certain streams and other features, like bi-directional streaming and stuff like that. Yes, we have to do L4 load balancing in AWS and do our own SSL termination there ourselves using Envoy.

Yeah, it's those nuances. They seem like the same thing, both cloud providers, but Kubernetes is just such a fantastic compute platform. It really spoils you when you have to dig into using a resource that is not cloud agnostic, like a load balancer. All these nuances show up and you have to work on top of that and abstract over them and fill in the little gaps here and there with your own setup.

**[00:22:49] JM:** You talked a little bit about the kinds of modulation of metrics and sampling and what you might want to store, like how many metrics you might want to be storing and how you can optimize for that. Can you tell me a little bit more about how that gets modulated? Let's say, I've given my developers free rein over how many metrics they want to store and how much data they're storing and then all of a sudden, my costs start to ramp up, do you just have some dial that they can tune to automatically constrain those in an intelligent way?

**[00:23:26] RS:** This is something that comes up again, when you're getting to that next level of scale with Prometheus in just metrics in general, whether you're using Prometheus, or Datadog, or another existing vendor. A lot of people don't give you any controls in this space. A lot of the time, it ends up being and I've seen this at Uber before we put in these controls and I've heard a lot of this as well from other end-user customers, where suddenly they're paying 20%, 30% more in their bill, just after 7, 14 days into a month, because of a deployment here, deployment there, a feature turned on here or there.

Really, they're at the mercy of going back to that team, asking that team, do they need that much? Asking them to go and spend engineering time to either be more cost-effective with the metrics they're using, or develop special configuration files that will let you drop some of those metrics. There's really no tools to actually tell you which ones are the ones that are hurting you.

You can perform certain queries. I can use the count Prometheus function to count how many time series are underlying behind a certain metric name and stuff like that.

How do you even know which metric names to be looking at? There's a few tools out there that a community-supported, but that none of them really let you get to the bottom of it quickly. We have a metrics profiler that we offer in our user interface that lets you actually view the top list, basically the metric names that are appearing the most frequently and hence, have them is underlying time series data underneath them.

We also let you actually see what also the current dataset you're viewing, what metric labels are on them, so basically the dimensions on the metric themselves, and what the unique cardinality values of those are. This lets you very quickly understand, “Oh, this dimension on this metric causes my entire thing to go boom.”

From the UI, you can also pivot and start to drop metrics that match a certain pattern on any one of those labels. Without any code changes, without having to go back to a team and ask them to do engineering work, without having to involve some SREs to look at some configuration files I might not know about, you can very quickly start to come back under budget and you can even have those predefined rules, like I told you about, where you may just stop dropping them immediately once you're over your budget.

You have these tools at your disposal to both rank priority of the metrics that are on the way coming in on the way into the manage platform. But also, to turn them off or to start to keep them, like a longer retention and resolution. I might ought to take some of my metrics and instead of keeping them for basically sampling them every 10 seconds and keeping a data point every 10 seconds, I might want a data point every 30 seconds, or 60 seconds, which would 3X or 6x reduce the rate of samples appearing.

There's those tools in our UI that are fundamental and there's also some other things we're looking at, like basically letting you cost back to your underlying teams. We may sync with your – you use the directory and if you're using SSO with us and do things like map certain metrics back to those teams. You could do cost accounting very clearly and delineate who is actually using all those metrics, which with all the vendors that we've used as these organizational features are not readily available.

We've lived the pain through this before, so we fully understand why these features are important and what people spend their time doing when they're inundated with the amount of data, or the cost, or the spend, or that thing.

**[00:27:39] JM:** The usage of both metrics and traces as you mentioned before you discussed this a little bit in the KubeCon talk that you gave. I'd love to know a little bit about tactics for how somebody using Prometheus, or just more specifically using M3, how can metrics and traces be used together?

**[00:28:06] RS:** I think what's super interesting here in the tracing space is you talk about sampling. Sampling is a really, really large problem. In fact, it's caused certain amounts of people to build infrastructure that tries to keep all the trace data for a 100% for a certain amount of time and let you see all the trace data. The reason why it's such an issue is that metrics gives you a volumetric view of things. You can see the P99, or your outliers and latency and that thing. That's great to get an idea at a high level of what's happening.

If you want to drill down into one of those 95th percentile requests look like, you're only going to get that if that trace – if there was a trace that fell into that bucket when it got samples. Even with the most sophisticated sampling algorithms out there and Uber developed some really interesting ones. Fundamentally, we can never guarantee you that you'll get a trace there. That's a large thing that requires something like tail-based sampling. Tail-based sampling is holding everything in memory, or it's in some place at full fidelity and then selecting the ones that are interesting to you at the end of a selection criteria.

You can imagine that for a minute, you hold all these four traces and then you work out okay, this trace actually filled into the 99th percentile, I want to keep that one. Tail-based sampling solves this problem, but it turns out tail-based sampling is really hard to do, not just for the infrastructure required to hold on to all those traces, but actually the selection criteria.

I just talked about how metrics gives you a grand idea of what the 99th percentile of a certain occurring request volume is, but you really want to link that in with the trace that fell into that bucket. We have this infrastructure name tree called the entry aggregator. That's something that none of the other open source projects in the space that are compatible with Prometheus provide at all. In fact, yeah, a lot of the metric vendors don't really have anything like this either.

It's basically a way that we do streaming computation on the metrics on the way in before they are going to our long-term M3 database. A lot of open source uses today, you stand up the database first and then they look at the aggregation tier, because it is another set of infrastructure you might need to run and if you don't run it yourself, Chronosphere is a great option for that.

Fundamentally, this aggregation tier because it can do computation on the way as metrics arrived in volumetric quantities, we can select okay, of this full set of operations we saw arrive in a 10-second window, or a 1-minute window, this one trace ID we know fell into the 99th percentile, so now we can pull that out of our tail-based holding pattern and select that trace for storage in M3.

That's we know we demoed. We have a demo of that that we built for KubeCon San Diego and we demoed that during our talk, so you can find the talk and look at the demo and there's actually an open source repository with the demo code in it. That's actually using a whole bunch of open source utilities along with Prometheus, but it's still hard for this to run yourself and that's why we are running it internally at Chronosphere for people.

It's a great thing to have, but it takes a fair amount of infrastructure set up, as well as configuration to pull off. Fundamentally, you need something that can tell you in real-time which thing to select. That works at scale. Most of these other existing solutions is all about flowing something into some huge database and doing some computation on later. That really doesn't work at scale. When you're talking about millions of samples already coming in to these pipelines, you need to do something like streaming aggregation, like the metrics aggregator. We found that to be useful for a whole bunch of other use cases as well.

**[00:32:23] JM:** There's a term ‘deep linking’. What does it mean to deeply link metrics and traces?

**[00:32:29] RS:** For many people, what linking metrics to logs into traces looks like is basically on a graph where you have existing metrics, if there's a container label, so some ID that represents the container name and a few other things, like the cluster it was in, etc., the linking that happens typically is that you take all of those labels that was on your metric and your dimensions, then you go search your tracing system and say, “Hey, around this time with these type of labels, do you have anything on the trait? Do you have any traces essentially?”

That's not great for a variety of reasons. For instance, this problem which we're talking about with P99 tail-based sampling, you won’t really know that a request that was processed by that container actually fell into the 99th percentile. That container might have processed a 1,000 requests and only one of them was the slowest. Doing this search by labels and matching traces with your metrics is not in our eyes linking at all. It's really correlation and it's a way of reaching into your bag of traces and hoping that it lines up.

It also relies on you to label and annotate your traces the exact same way that you're doing your metrics, which depending on what you're using won’t always happen. What we do with our deep linking is actually similar to what Google is doing with monarch. This is only available to developers inside of Google, unfortunately. Monarch is a system that basically puts the IDs of the certain traces next to their metric values. This is because monarch has a streaming aggregation metrics aggregator similar to M3.

Essentially, what ends up happening is you send all the trace IDs along with all the metrics. Every single metric you increment, you're sending the corresponding trace ID to an aggregation layer. Then an aggregation layer when it prints off, okay, this was the value for the 99th percentile for this 10-second time window, it chooses just one trace ID to correspond and make sure that it is written to your time series database next to it.

Now when you have a data point that you're viewing in a graph, you can click on that data point and it actually has basically the UID of the trace, or other unique identifier, depending on how you're identifying traces. You can go in oh, one time to that trace in the other system. It's not searching by labels. It's not trying to do fuzzy matching on the metadata of the metrics for us, the trace data. It literally will just go, here's the exact trace I want to see, and pull that up in your trace provider. That's what we're talking about there with deep linking.

**[00:35:30] JM:** You mentioned monarch from Google. Are there any other lessons from monitoring at Google that you've been able to import into Chronosphere, or just maybe monitoring from even Facebook, I don't know, conversations you've had with some of these other advanced companies.

**[00:35:52] RS:** Yeah, most definitely. I mean, when we were developing M3, we caught up with the gorilla developers at Facebook. We developed M3Ts.ED, which is very similar to Ts.ED, which is essentially an extra special fluid compression. You can get 11X compression ratios on float data or using Ts.ED. We're very thankful that exist. That was a huge breakthrough. I think the Facebook engineers deserve a huge credit for that.

What's interesting now though is that apparently, these systems are all in memory, whereas M3 and Uber was very expensive to run these in memory. We very quickly pivoted to trying to store them on disk. We needed to shrink our overall spend and infrastructure to a small number, because we needed to use our hardware effectively. Anyway, so we used to catch up with the Facebook folks. We also caught up with folks at Netflix and the Atlas engineers. Actually funny, you might mention monarch. Actually, one of the co-authors of M3DB with me, Shi Chen, he actually works on monarch as one of the most senior engineers now at Google. Not everyone left to work on Chronosphere. There's some that are is still there doing great things with M3 and Uber and there's also people like Shi that are now using his skills and great talent to make monarch better at Google.

Yeah. We pulled on a lot of different things. We catch up actually still regularly with some of the engineers at the companies I just mentioned. We keep in touch. It's a pity that in some ways that we're the first ones to find the time to actually open source this project in a way that's consumable and others can use outside of a tech company.

We're just glad to be given the opportunity and I never really thought that we were scaling the solution for Uber that there would be launch payment companies, or what else have you, role running this code now. It's definitely been quite a transition, but it's exciting. I think that every time I wake up and I have Slack messages or tweets telling me that someone has a small issue with M3 and that love – they just need a little bit of help, or they want to tell me how – I love criticism, so I like them telling me what sucks about M3 for their use case.

It really is refreshing to be like, “Hey, this is something that's going out into the universe.” That is letting people solve the same problem in a similar way that we can all contribute to, rather than these different solutions that are glued and/or sometimes together at different proprietary lands and that's really hard to pace together systems.

I am all about engineering. My granddad was a civil engineer. I love engineering. I think that we need more primitives for what we're doing today. I think that things like M3 and open metrics, which I'm a contributor to just really lay this groundwork for hey, how do we build and monitor systems? How do we write software that can be monitored? How do we build rollback systems and order rollback our deployments, so we can move fast, break things, but then immediately recover?

I think it's really what we're doing here, not just us, with the whole open source community is doing and what everything else is happening in this monitoring space is changing the way that we develop software. We can monitor things we never could monitor before. We monitor first, think later. It's not like a black – when something goes wrong, because we monitor first and look at the data later, because we have the tools and the cost-effective ability to do that, it's not like we're poking around in a dark room anymore. We can see what's in the room.

The fundamental changes, we talked a lot, we deep dived down a lot of topics here, but what at the meta-level is happening is just software's becoming way more of a white box, or an open box and transparent box that we can see. That's fantastic.

**[00:40:16] JM:** Cool. Well, Rob. Thank you so much for coming back on the show. It's always a pleasure to catch up.

**[00:40:21] RS:** Yeah, most definitely. It’s fantastic for being here and thank you so much for the opportunity to speak again. Yeah, it's been – we've definitely had a busy time since we last caught up, so it was great to get a chance to see you're doing well and hopefully, one of these times again we might run into each at a conference, but probably won't be any time too soon, I suppose.

**[00:40:44] JM:** All right, Rob. Good talking.

[END]