EPISODE 767

[INTRODUCTION]

[00:00:00] JM: Uber manages the car rides for millions of people. The Uber system must remain operational 24/7 and the app involves financial transactions and the safety of passengers. Uber infrastructure runs across thousands of server instances and produces terabytes of monitoring data. The monitoring data is used to understand the health of the software systems as well as relevant business metrics, just driver efficiency, and daily revenue, and user satisfaction. Uber adapted the Prometheus monitoring system to manage their monitoring data. Prometheus regularly scrapes metrics across infrastructure to gather time series data about the state of everything across Uber.

As the usage of Prometheus has grown within the company, Uber has had to figure out how to scale their monitoring platform. M3 is a monitoring system built at Uber to scale Prometheus and to provide a platform that can effectively scale the data storage as well as the query serving.

Rob Skillington is a staff software engineer at Uber and he joins the show to talk about monitoring within Uber from the requirements of the system to the implementation of M3. This was a deeply technical show, and also went to the higher levels of the importance of monitoring. It was really fun to have Rob on the show and we barely scratched the surface of what I would have liked to get to.

At Uber, M3 powers the dashboards, the ad hoc queries, the alerting. M3 was open sourced to give other users access to a scalable Prometheus solution. In a previous episode with Bryan Boreham, we talked about one strategy for scaling Prometheus, and today's episode covers another scalability solution with M3.

We have a show coming up soon that covers monitoring at LinkedIn and it's very interesting to see how important monitoring is within an organization. You think of monitoring as perhaps something that only infrastructure people care about or only people that are issuing alerts and doing things that are close to the software engineering, but monitoring really is the life blood of

how we understand the metrics within a company. It's like the health of a company. Now always super closely tied to what you see on a company's balance sheet, but sometimes it is closely tied to what you see in a company's balance sheet. So there is a deep connection between monitoring and the business health of a company. I hope you enjoy this episode.

[SPONSOR MESSAGE]

[00:02:48] JM: Hired simplifies the job search for engineers with a data-driven personalized matching process. Head to hired.com/sedaily and create a profile today. By creating one profile, you'll be matched with over 10,000 companies looking for engineers like you. Hired uses intelligent matching technology. Data science with years of experience matching engineers with jobs, and Hired also has a human in the loop.

Hired gives personalized career coaching to match you with opportunities based on your skills, industry, interests and desired salary. Create a profile today at hired.com/sedaily. Find a job that you truly love that is personalized to your background and your preference, and if you aren't an engineer, Hired also helps designers, engineering managers, product managers and other tech workers find their dream jobs. Just to hired.com/sedaily and check it out.

[INTERVIEW]

[00:04:03] JM: Rob Skillington, you are a staff software engineer with Uber. Welcome to Software Engineering Daily.

[00:04:09] RS: Thanks, Jeff. It's great to be here.

[00:04:10] JM: You've been at Uber for 4-1/2 years. The first show that we did about Uber was about three years ago and we had Matt Ranney on the show. He talked about how fast the number of services at Uber had grown. So Uber was growing rapidly on all dimensions. There was the number of rides that was growing, the number of engineers that were working on the system. So the overall throughput of Uber was constantly being tested at that time, and you were at Uber at that time. What was it like being at Uber back in 2014 when you joined?

[00:04:46] RS: Yeah, back in 2014, it was moving very dynamically. I remember joining the marketplace team, which handles the real-time matching of riders and drivers and we were growing from like 10 team members to 30 or 40 within the space of one to three months. It was a very interesting time. I think there was a lot of evolutions that needed to take place and many different teams had to radically change how they approached problems, what their opinions of the software and how software should be written and released had to change to keep up even with business itself.

[00:05:31] JM: So that time back in 2014, you were working on this team that was focused on dispatching services and the lifecycle of a trip. This is core service infrastructure. You did get to take advantage of some of the platform infrastructure that was getting built at Uber around that time. So I imagine there were some monitoring solutions and deployment solutions that were widely used throughout the company. To what extent was the "platform" engineering at Uber was standardized across the organization?

[00:06:07] RS: Yeah, there were several pieces that were more self-service than others and some that were true platforms as opposed to basically pretty simple pieces of infrastructure that required a puppet change. So the deployment system is probably been the only system at Uber that was self-service back then and remains in a relatively similar form today to in terms of the user experience of how it worked back then.

Metrics was also similar in nature. However, the onboarding and the self-service of it was not really nearly as proficient and scalable as it is today. However, it was at least something that came for free. Then there was other parts of infrastructure like adding capacity, using HAProxy to forward requests between different services and that service discover layer which was heavily based on puppet. Another database we used called Cluster, which actually was primarily supposed to be used for networking and provisioning service and keeping an asset catalog, but became kind of a database that was used for HAProxy when it was configured to workout, which backends to route to and basically became a very critical piece of infrastructure that was used in very freeform manners, which actually made it relatively difficult, because only SREs could really understand how to add capacity to your service. How to help you route to different services and onboard you on to things like Redis and MySQL and things like that. Really, back

in the day, all we had that was self-service was metrics and deployment tools and everything else was kind of handled by SREs.

[00:08:07] JM: So there was an extent even back in 2014 to the automation of a monitoring platform. So if you were building an in-house service like calculating prices, or having the lifecycle of a trip, or doing dispatching for a ride to a certain user. If you're building these user level services, there is some internal tooling that you can just plugin to. It works really well for the Uber platform, but overtime as Uber was growing and diversifying into different services and the other aspects of engineering were changing, that monitoring infrastructure started to fall over is the sense that I get.

What were the strains on Uber's engineer architecture that led to the monitoring not being able to keep up with the demands?

[00:09:02] RS: Yeah. So it's actually a pretty interesting story. Because the monitoring platform was relatively self-service, a lot of people kind of on-boarded on to it and weren't really, I guess, cognizant of how it operated or what kind of strain to put on different parts of the system, and we were using StatsD back then with a whole bunch of different StatsD clients, and some of these StatsD clients, especially the community released ones, didn't really do a good job of offering the data they put into the UDP Packets that they sent to the StasD daemon.

We were actually migrating from one data center host to another at that time and we found that actually the strain that we put on the routing infrastructure just of like the switches themselves caused a lot of false starts with that new data center migration. So we had to be very, very careful about this migration and it ended up being a very critical thing that kind of began – Made it very difficult essentially to move into this new data center because of the high packet rate of UDP packet sent by the StatsD clients.

So basically those are a bunch of efforts back then to at least modify the StatsD clients and get us out of a short term kind of problem there just by basically buffering the data on the client side. Then kind of like once we got past that hurdle, there was the obvious strain of just the metrics platform not being able to add capacity to it and to be able to keep people's metrics as we needed to add capacity to the system. Whereas previously, there was such challenges, like it

was very difficult to keep data when we basically needed to add machines to the metrics platform itself.

[00:10:58] JM: Okay. So when the metrics platform needed to scale up from what I'm hearing, it was hard to retain data in that span of time when you were doing a scale up. Is that right? Is that what I'm hearing?

[00:11:13] RS: Yeah, it was relatively – It took a really long time. It caused a lot of pain, because alerts would go off and people would get paged, and it just wasn't really scalable just in terms of both the software solution itself and the process around providing this to people in a way that was not going to disturb them and could be trusted to wake them up at the right time.

[00:11:37] JM: So you were on this dispatching team in 2014 through your first – I don't know, year or two at Uber, and eventually you transferred to the metrics infrastructure team. So there was something about metrics infrastructure or platform engineering that was interesting to you that made you want to switch from kind of a product-facing service to more of an infrastructure problem. What was it about infrastructure and platform engineering that made you want to move over to that side of the house?

[00:12:12] RS: Yeah, it was multi-faceted. I think that there were several times during my time developing services and backend application in marketplace when I noticed that whether something worked well or didn't worked well on the infrastructure level made a huge difference to how well my service ran and how easy it was for me to deliver on the promises I was making to build services for marketplace and the reliability it came with.

For instance, I remember several experience where just being able to see in very high resolution data at like 10 second intervals how the file descriptors of my service were fluctuating. How different application level statistics around how many users were going down this code path versus another code path and what was the breakdown of the users going down some of those code paths where they're using this client version on the mobile app or another really made a difference. I think that that's probably a pretty recent change to software development. We didn't used to have this level of introspection and insight, and I could tell that this part of infrastructure was going to change radically and fundamentally over the upcoming

few years, and I knew at Uber it was going to be absolutely essential to stay ahead of that curve and be able to offer really scalable tooling for engineers to be able to scale their systems and stay ahead of the curve. So that's really what led me to the platform and infrastructure team.

[00:14:02] JM: This is one of the cool things about working at a gigantic company, is you will just wake up one day and there's an email in your inbox that's like you're job just got a lot easier because somebody on the platform engineering team built a new tool and all you have to do is upgrade your package and you now have access to that tool.

I can imagine being on the other side of that and building the tool that helps thousands of colleagues that are working in engineering and then all the downstream positive impacts of that on the customers. I can imagine it being a desirable place to be in an organization.

[00:14:43] RS: Yeah. I think that there was definitely an air of, "Well, let's just reimagine how infrastructure and the parts of infrastructure that we want to develop and how we're going to deliver that to the engineers," could be different, could be very different this time to how it was at previous companies, how bringing a lot of like the shiny, say, features from a lot of the community-driven software to really large scale companies was a very exciting prospect.

I'd worked at Microsoft before Uber and various other companies and really there's a lot of things that I would have loved to have used at my time at my Microsoft, but I really couldn't, because the level and the scale at which these tools and platforms need to operate at traditionally doesn't mean you can take a community-built piece of software or a piece of infrastructure and just it at a scale 100X larger than what was intended.

At Uber, I think, as an infrastructure engineer, it was really exciting. We knew that we could merge kind of like the ideas of the old and the new and really rethink how infrastructure could be delivered at a company like Uber.

[00:16:06] JM: Just a little more on the platform engineering before we'll get into Prometheus and monitoring and some more stuff that is a little bit lower level, but on the platform engineering side of things, there might be people listening who are wondering, "Okay, I can use AWS today. AWS has a bajillion services. They're really well-integrated with each other."

There's also tons of open source software. The Kubernetes community is moving really fast and there's also the Apache projects, like Kafka and whatnot. There are tons of tooling that's out there. Why does a company like Uber or Netflix or Airbnb need a platform engineering team to build stuff customized to their in-house needs? Why isn't this stuff just all taken care of by AWS or the open source world or whatever other off-the-shelf software or vendors? There are tons of vendors out there. Why can't I get what I need from name your monitoring company?

[00:17:10] RS: Yeah, it's a great question. I think that to some extent we are actually seeing that it is a bit of both. Most companies like Uber and other companies at large scale are actually trying to use things that are off-the-shelf as much as possible. However, we do prefer to use open source software that we can deeply understand and then if necessary make changes to as opposed to proprietary off-the-shelf software, which if we outgrow it, then it is really, really difficult to change and modify and keep up-to-date with the needs that we would have.

So I think it is becoming more complex. We had an early Google engineer come and talk to us a few years ago about how at Google like that was a lot of their challenges they were solving these infrastructure problems. No one had ever done it before. They needed to build it and they built it all in-house. He said, "Today, I would hope that companies like Uber and such, you don't have to do as much of that."

Yeah, and what he's saying is true in some ways. We don't have to solve as much as that, and the qualifier as much though is the important part. There are still significant, not necessarily deficiencies, but like just shortcomings in some of the scale at which some of these open source projects can deliver for companies.

However, I do think that it is 100% getting better. Stuff like Kubernetes can scale to thousands of instances now. Back when Kubernetes got started, it could only scale to 500 instances, which is – But now it can do thousands. So I think it's an evolving landscape and companies will sometimes still need to choose challenges that they need to solve themselves or at least combine with an open source project and build on top of. However, it is getting better by the day.

[00:19:16] JM: Right. I mean, you certainly see continued "infrastructure challenges", but the ones that Google was solving were things where you had to build custom hardware. You've got to go down into the lower level guts of operating system code. You have to rewrite the operating system, whereas today like Stripe has to build infrastructure for wiring payments systems together. Yeah, that's hard, but at a certain level it's not as so crushingly hard as programming hardware. Also, it's kind of at a higher level of the stack. So in some sense maybe you couldn't create more value. Anyway, the challenges are different. I think that's for a different show.

Let's get into talking about monitoring. So Prometheus is a monitoring tool we've done some shows about. Explain what Prometheus is and how Prometheus interacts with infrastructure that it's monitoring.

[00:20:18] RS: Yeah. So Prometheus is a fantastic project. It really is the MySQL of monitoring when I think about it, and that kind of analogy is good as well, because MySQL is a great database and powers so many things in the world. But when you think about how it scales, it still has a single write master. I'm sure you can do your own layering on top and kind of try to avoid that issue with it, but at its heart, its' really like a single database node that takes the mutable data and allows you to mutate that.

Prometheus is similar in that. It's a fundamental building block that allows you to basically store a whole bunch of monitoring data in a single server, and the way in which it does that is a great building block. For a lot of companies and organizations and developers of anything, that's solves a lot of the – They're getting started and a lot of cases the continued support for basically being able to monitor and kind of understand how their software is operating.

Similar to the metrics platform, at Uber it has multi-dimensional metrics now, which just fundamentally make it so much easier to be able to slice and dice categories of users or categories of your hardware and how they operate. So I think it's a very important piece of software infrastructure for everyone out there in the industry and it's really making a difference, I think, being able to get started with such – Like a performant, powerful software means that you don't have to basically a whole lot of time yourself setting up your own monitoring – Connecting all these pipelines. That's the other thing that's great about Prometheus is say single server that

handles the ingestion, the alerting and also even a dashboard for just like splunking around in some of your data.

[SPONSOR MESSAGE]

[00:22:37] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:24:45] JM: I love the comparison to MySQL, because MySQL is a time-tested, well-proven database, but as we're going to get into scaling MySQL, it has its own challenges. You'd

probably don't want to be responsible for scaling a MySQL database unless you're an expert in scaling MySQL databases.

[00:25:07] RS: Yeah, most definitely.

[00:25:08] JM: And that's kind of where we get to with the conversation of Prometheus at Uber. So just to give people an overview for at least how I understand Prometheus, you have Prometheus on a single server. It's periodically scraping other services and things in your infrastructure to collect metrics. It's gathering in those metrics. There's an ingestion process for prepping those metrics to be ready to store, and then you store the data on a single server, on the same server, and then you can query that data from the same server, and all the stuff is taken care of by Prometheus. But where this starts to fall over is you have questions like, "Okay. So I've got a Prometheus instance. Should it scape all the services at my company? What happens if I have so many services that this single Prometheus instance can't handle it? But then does that mean I'm storing my data all over the place?" Then there's a question of like, "How do I shard the database that I'm storing all my Prometheus metrics in?" I guess talk a little bit more about what Prometheus is at maybe a small scale and what it starts to look like at a bigger scale?

[00:26:22] RS: Yeah, most definitely. So your Prometheus on the small scale I think is really where the sweet spot is. You have basically the ability to scape all of your services and not really have to worry too much about complex configuration. You can use Grafana out of the box with a single data source. If you want high availability, you add a second Prometheus server and you basically scape from both of them in an active, active way. Then in Grafana you kind of just choose which one you want to actually view the data from.

So that kind of like works quite well for people while everything can fit on a single Prometheus server. There is some more interesting parts to it where if you're kind of in a more ephemeral world, like the cloud, and that instance tends to disappear or could be restarted and maybe the disk is not accessible. It basically means that you have to think a little bit more about like how to backup this data, how to save it.

I think with Prometheus, it's really about the simplicity model and they're not trying to solve if you really care about data durability and the kind of different retentions and any down-sampling you might want to go with, they're not trying to solve these problems. So I think it's really great at the beginning. It gets you started quickly. It can scale quite well for many organizations, because it can do like 400,000 samples per second on a single machine, which for a lot of cases is probably enough. Although that's talking about a pretty expensive server to achieve those numbers, but it's really great to get started with.

So when you think about how Prometheus runs when you've got so many servers and applications and pieces of infrastructure that you're monitoring that a single Prometheus server can't really solve the problem for you and you have to start adding more and basically dissecting the data. That's where it definitely gets a lot more complex. You can start to take like human ways of sharding it, which is like I have this application stuck to this Prometheus server. I have this other application stuck to another Prometheus server. But I think that, again, that could work in the initial days when you need a little bit more scale, but it starts to fall apart pretty quickly, because then in Grafana, you have to have some knowledge of this mapping or build some complex system to do this mapping for you. Not just that, you find that if there's any application that ever overwhelms a single Prometheus server, then you kind of out of luck and you need to really think a little bit harder. That's where most metric systems that want to allow you to grow to a reasonable size more than one or two servers will actually start to shard the underlying data using like a hash function so that it's like well-distributed in a very even manner.

[00:29:24] JM: When you get this sprawl of different instances of Prometheus that are maybe arranged differently, like maybe you have – I'm just speculating here, but maybe you have a team and everybody in the team uses the same Prometheus instance and it's just gathering a bunch of metrics and then the entire team, the sum two pizza team within Uber has a collection of dashboards and anybody in the company can login to those dashboards and look at them, but then maybe you have some other team where every single service within that team has its own Prometheus instance and then each of those Prometheus instances has a collection of dashboards. There may be like uneven kind of file structures or addressing structures for the different dashboards. Is that realistic? Does that happen?

[00:30:15] RS: Yeah, definitely. So you can imagine – We've got 4,000 microservices at Uber. Even if you say you group like 10 together, that's still 400 data sources in Grafana and a big dropdown, and you've kind of got to like work out where yours is going to be. Also, you kind of have an outage of like one of the single instances of those Prometheus service, you need to know which one is a backup and when to look at the backup versus looking at the non-backup.

It starts to get unwieldy pretty quickly. There's also some things. At Uber we do like install automatic alerts and kind of discover which metrics you're using so it can like workout whether you need to have like a MySQL slow query alert or a Cassandra alert with respect to like how well replication is keeping up and stuff like that for cross data center replication.

That model breaks down if you have to do a for loop of 400 Prometheus servers and look in every single one to see if a team or a service may add some metrics. So it starts to become really important to have a central place that really has all the data together so you can operate in a much smarter way.

[00:31:33] JM: And the matter of data management seems pretty core to what you ended up doing to scaling Prometheus. So Prometheus has its own time series data format. Again, the traditional Prometheus way is just written to the same server that is doing all the other work. Tell me about – Well, actually we should start with where Prometheus fell over, I guess, because Prometheus is only like – I don't know, three, four-years-old, something like that, maybe two-years-old, and I assume that there were some period of time where Uber started using Prometheus or it was like lumpy adaption throughout the company and then people started adapting it more and more and then people were starting to have issues with it. Maybe you can get into the issues about like where Prometheus really started to fall over within Uber and when people at Uber started putting together a team to say, "Okay, we got to rebuilt our whole metrics platform."

[00:32:36] RS: Yeah, definitely. I think that it was a multi-faceted, again, kind of challenge. Like we had a lot of people using StatsD and Graphite still when we started building M3DB and M3, and we also, yeah, some teams using Prometheus and they were – Honestly, everything works pretty well on a small scale, and then the teams that were basically running these Prometheus instance honestly just didn't launch to basically pay the headcount in terms of SRE's to continue

to manage the complex configuration. Like they had like – Once they got to about 6 or 8 Prometheus servers, the amount of git config, the amount of puppet, the amount of basic like really complex configuration files of Grafana and kind of the templating of your dashboards and like where one dashboard mapped to which Grafana data source, because the dashboards were built by source control was starting to get really out of hand.

At that point you've got to make a decision on whether do you double down on basically doing this mapping using source control and SREs and really being very process-driven and strict about how you run the installation, or do you kind of lean more on a centralized platform that while itself is complex under the hood, at least look simple to operators. That I think is where it started to become like a really good idea just to begin to actually build technology rather than process and configuration management solutions to solve the problem.

[00:34:29] JM: M3 is this metrics platform that you've worked on at Uber. Explain what M3 is.

[00:34:37] RS: Yeah. So in a nutshell, M3 is a scalable metrics platform. It tries to offer us basically the similar kind of data model that Prometheus does, and soon we're adding Graphite support in open source. We actually internally have that at Uber. It's just part of – Not part of the open source code base.

So, yeah, it really at its heart is trying to basically make it simple to have extremely high cardinality metrics data for engineers to be able to both leverage and also kind of interact with for learning purposes and for like building auto-scaling rules and auto-balancing rules for instance. When you use a cloud vendor, you can set up all these rules that like trigger other things, that scale up, scale down things. Similarly at Uber, we kind of do similar things where we will shift like the mapping or some workload between clusters based on stats. We also use it for like monitoring SLAs between services. We use it for basically looking like of a data center level of like how is our data center actually performing in the networks, which level and how warm some of the pieces of hardware? We do a whole lot of other – We basically use it for pretty much anything that we want to measure overtime.

[00:36:01] JM: What problems does M3 solve for service owners at Uber?

[00:36:06] RS: Yeah, M3 for service owners is really about, A, working at what you depend on and how those things are performing. So if you use Cassandra or Redis or any other piece of storage technology or cash layer, and you also want to kind of understand like how in your application is actually performing in terms of a CPU usage, in terms of RAM, in terms of network in and network out, you can get all that data. So you get a really good idea of basically how you affect others and how much resource you're consuming. If you're using some of these dependencies, like storage, basically what kind of performance are you seeing from those? That can help you drill down basically and workout like when your application is performing poorly.

Then there's the other side of it, which is I think actually where it gets more interesting and in the past hasn't been basically used as widely is where you actually start to instrument your code paths very delicately to workout in business terms what your application is doing. So you can use gauges and counters and also latency measurements to workout, basically how your code paths are performing, and you can even do things like working out, "Okay, how many, for instance, trips are beginning or ending from an airport, or in a geo-hexagon, like what is the surge overtime and how to basically workout whether that's performing well or not."

So it's kind of at the application level, it's starting to become I think with Prometheus and M3 and other things that allow for high-cardinality data. Developers and engineers are really starting to get very meticulous and specific data back from how their application is performing.

[00:38:08] JM: So this is not just about I need to know if my service is up or not, or I need to know if I need to scale up my MySQL cluster, because it's getting hammered by requests. This is like translating data of various parts of the stack into business value by having better dashboards, better visualizations that can be used by people throughout the organizations. There's a really wide range of value that you can get out of improving the monitoring platform, which includes the data storage layer.

So let's talk about the data storage layer, because you built a new storage backend when you built M3. So this is M3DB. Explain what were the constraints on the previous data infrastructure that was underlying monitoring at Uber and why did you need to build a storage backend?

[00:39:10] RS: Yeah. Once we solved the original challenges of being able to basically add capacity in a way that didn't lose data and was up all the time and was highly durable, which we actually used Cassandra for originally. That was a great milestone and it was horizontally scalable, and I'm putting air quotes around that, because while that is 100% true, it isn't always cost effective to horizontally scale if your scale is at a rate of scale where the cost is just too difficult to maintain to receive the benefit from it.

Basically, what I'm talking about here is that you start to reach a point where just adding more on this fixed linear scale starts to reach some very high numbers. For instance, we're buying millions of dollars of hardware for Cassandra when we knew we needed to buy much, much less if we were going to be able to keep the costs of monitoring at Uber to a few percent of the entire hardware cost. We now have a much better cost curve, thanks to M3, and it kind of lets us add really high-cardinality to data. So we have like 9 billion metric IDs and we still say pretty very lean and kind of like offer like a very good cost structure for the business so that even though we have all these data, it's just extremely highly compressed.

So with M3 TSA compression we get about 7.6% – Sorry, 7.6 times the order of compression with just storing if we were to store like the data in 64s for the timestamp and the value, and that's on our data. We've actually seen compression ratios of up to 11X and larger on different datasets. That really enables us to store that much data in a way that's not going to impact the business and basically cause us to use like 5% or 10% of the entire compute capacity and hardware just to serve monitoring data.

So I think that was a really big differential, and like you look at Prometheus, Prometheus does the same thing actually, and that's why it is great for small shops, because even with a single server you can get quite a bit of runway because it has such high-compression and because the index doesn't take a ridiculous of size. So it's really from continuing to develop the efficiency on the storage side and the compression and also how the index is kept that we can provide this outsize benefit for a relatively small amount of hardware.

[SPONSOR MESSAGE]

[00:42:15] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly. You can store and manage unlimited data, you can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your sites functionality using Wix Code APIs and your own JavaScript. You don't need HTML or CSS.

With Wix codes, built-in database and IDE, you've got one click deployment that instantly updates all the content on your site and everything is SEO friendly. What about security and hosting and maintenance? Wix has you covered, so you can spend more time focusing on yourself and your clients.

If you're not a developer, it's not a problem. There's plenty that you can do without writing a lot of code, although of course if you are a developer, then you can do much more. You can explore all the resources on the Wix Code's site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix Code experts.

Check it out for yourself at wicks.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you can do with Wix Code today.

[INTERVIEW CONTINUED]

[00:44:13] JM: To come back to the thing we discussed earlier with like who needs platform engineering and why do they need it, isn't the problem of time series database worked on a bunch of people who are – Like their existence is tied to them making a good time series database. There's InfluxDB, there's TimescaleDB, there's a bunch of different time series database. Why did you need to build one yourself?

[00:44:39] RS: Yeah, that's a great question. So I think there is a bunch of them out there. They're not always as scalable as you may think especially in terms of the scale that Uber that

needs to run at. I think that we just found when we tested these solutions that – And the other thing is while some of them do scale, the cost structure again is not just at a place that would put us less than 10% of our overall hardware usage, which when you talk about monitoring, you really want to keep it to 1% to 2% of your overall spend, maybe 3 or 4 if you're pretty hungry for metrics. Because otherwise it just an extremely high overhead.

It's really a mixture of both the scalability of it and cost structure of like how expensive is it for each node, and we found that, yeah, just none of the existing databases out there, at least in terms of storing billions of metrics was really going to keep our cost structure in a way that allowed us to store as much data that we're storing today without consuming really high numbers of our overall hardware spend to keep that data and operate on that data.

So that's really what it's about. I think even though our previous solution with Cassandra was fine for us to horizontally scale out, but just the numbers and the cost of which it would have been to the business meant that we would have had to clampdown on developers way harder. When you're talking about basically R&D and how fast you're going to integrate on your R&D, I think it's really important to be able to kind of just foster this idea that you can just monitor and measure anything. Otherwise, yeah, you need to be much more careful – And the onus is kind of a new one on how do we even begin to work out how to measure this and do things in a safe way.

[00:46:39] JM: Did you say that 1% to 2% of an organization's entire spend is in – Like the entire spend of Uber should be devoted to monitoring?

[00:46:48] RS: So I'm saying that in some companies, that is definitely what could be – Yeah, the kind of spend that you could plan for. Because when you're thinking about like hardware itself and the applications that run, you can't do it in a reliable way at a huge amount of scale unless you're dedicating some percent of your overall hardware to actually reliably insure that things don't get wrong without you knowing about it.

[00:47:15] JM: Okay, very interesting. Let's get through a little bit more of the architecture of what you've built at M3. So you have the M3 coordinator. So this is a sidecar to Prometheus instances. So I'm guessing you deploy your Prometheus instances to Kubernetes or something

like a Kubernetes, and you have a container model where you have sidecars, and the sidecar container of the M3 coordinator can't perform things that are sort of additive to Prometheus, because M3 is expensive to Prometheus. So the M3 coordinator servers as this global query and storage interface. So you have the sidecar pattern and then you probably have some kind of orchestration that you can do between those different sidecars or the different M3 coordinators that are running alongside those different Prometheus instances. Can you talk a little bit more about the role the M3 coordinator plays and how that relates to the M3 database.

[00:48:18] RS: Yeah, definitely. So the coordinator is a really easy way to use M3 with Prometheus, and the reason that it plays such an important role is that it not only basically ships your metrics to M3DB from Prometheus, but it also down-samples them if you're keeping metrics at different granularity and retentions.

So for instance, I might keep, say, all of my Prometheus data for 30 days without any down-sampling at all. So if you're collecting data at 15 seconds or a minute, you might just say – In Prometheus itself, you might just basically ship that to M3DB without down-sampling that at all and keep that for 30 days. But when you start to talk about a bunch of longer resolutions, for instance, like you're talking about keeping data for 3 months, 6 months, a year, that's when you really can't afford to keep it at the granularity of 10 seconds or 60 seconds most of the time, because when you go and query for that data, it's just going to take such a long time to actually get all the data points together, sum them up together or sample them in a way that you're going to be able to visualize that. Otherwise you're sending back just hundreds of megabytes of data to the browser, which is not really going to work at all.

So the coordinator plays an important role of essentially down-sampling that data in real-time. So other Prometheus long-term stores, like Thanos, they do something a little bit differently where they'll basically take the Prometheus data that basically hasn't been down-sampled at all, put that in the cloud, and then they have this offline process to go and take that raw data, down-sample it into, say, a 5-minute tile, a 10-minute tile, and then write it back to the cloud.

We just found a scale of that model of basically writing it lower and then basically having an offline process to compact, pull the data back down and then push it back up just in terms of like network, bandwidth, RPC, serialization, deserialization and the whole process and monitoring

that process and making sure that process if working reliably was just too difficult and costly for us. That's why we always down-sample this data on the way into the database and why it's required to have one coordinator per Prometheus server, or you could go the other route and have one coordinator for many Prometheus severs. But because it's doing down-sampling, that needs to be just one of them. And we have a solution if you up grow like a single coordinator instance. You can run a clustered aggregation server that we also have in open source now, but it's a lot easier to get started just using the coordinator.

[00:51:14] JM: Okay. I want to describe the conversation we've been having in summary and then we can zoom out a little bit and just close off. Tell me if this reflects Uber correctly. So you got the point where you wanted to use Prometheus. Prometheus, because it is a single server and its strategy of just scraping a bunch of metrics from different services, aggregating those metrics, writing them to the same single serve. That is not a scalable enough solution for Uber.

So you had a couple of different scalability complements that you added to Prometheus. One of which is an entire database, time series database, that makes querying and the storage of data in a way that can be queries via PromQL, the Prometheus query language, it stores it more efficiently so that Uber can save some money. You've also got the sidecar model. So you have lots of Prometheus instances, these Prometheus instances are scraping these metrics and the sidecar model for each of these Prometheus instances, which is the M3 coordinator. This M333 coordinator sidecar does just things that are very useful to each of these running Prometheus instances, like as the Prometheus instances are scraping the data and gathering the data at a certain frequency, you can have something on the sidecar that is, for example, down-sampling that says, "Okay, Prometheus, I know that you scrape the data ever 1,000 milliseconds, but we're going to down-sample that and just take it from every 1,000 milliseconds to ever 5,000 milliseconds, and then we'll store that, because we don't want to store all of that data that you're scraping."

Then there's some other features that you get out of having a sidecar, but this is the overall picture of where you have taken M3 to get a scalable version of Prometheus. Is there anything else that you'd like to kind of add around what value M3 is providing to Uber before kind of zoom out and close off?

[00:53:27] RS: The value that M3 really brings to Uber is safety alongside speed of development. With M3 and ability to really store high-cardinality data, we're able to measure things much more than we were previously able to at an unprecedented rate and we're able to develop a lot of platforms on top of it. For instance, like we have auto-rollback mechanisms. So we can kind of like install the standard alerts and procedures that basically when you deploy a service at Uber and it's not functioning the way that it usually does during a deployment, we can automatically roll it back.

It's kind of like this notion of like Facebook first said like move fast, break things. We're wanting to get to a point where we move fast but don't break things and automatically can avoid breakages in a way that still allows us to write and develop code and kind of ship features in a fast way.

So I think it's really like safety, reliability, a platform to be able to actually do things like anomaly detection on at scale, which again having a centralized store is just really important. Otherwise, the discover of all these stuff becomes so much harder and which parts of which system data is living in which data store. It just becomes really difficult to build platforms on top of that kind of model.

[00:54:56] JM: Cool. Okay. Well, there's a lot of other stuff we could have discussed, but I know we're out of time. A recent show that we did was on Cortex, which is an another way of scaling Prometheus. That's a little bit less fully-fledged, like they're not rewriting a database layer, and I've seen other approaches to scaling Prometheus.

Tell me about any open source adaption you've seen of M3 or sort of where you think the project is going to go. Do you think this is domain-specific to Uber or are other companies adapting it and will they want to adapt it?

[00:55:30] RS: Yeah, definitely. Yeah, I think when you talk about, yeah, other kind of like horizontally scalable metrics platforms, like Cortex and stuff like that, what's really important is the cost model, how much is this costing you to store this much data. Netflix first developed Atlas, which was like a scalable metrics platform for them, and the jerk that kind of gets passed around is like Netflix is not a TV and film streaming platform. It's a monitoring platform, because

like that's how much the cost of monitoring at the scale that they needed to do it was costing them at the time with Atlas.

I think that like they've obviously made improvements to Atlas, and similarly Facebook have something called Beringei, which was based on the Gorilla paper. But all that data is basically in-memory in those systems, which means you just can't pack those service nearly as densely as you can with systems such as M3.

So, yeah, it's definitely not the main specific. It should be able to be used at a whole bunch of different companies. We are seeing definitely some companies at scale that are beginning to use M3, which I'm very excited about. I can't really mention them without any kind of legal sign off from their side. But I'm really excited about hopefully other companies won't have to build their own solutions like this again with M3 being out there. We're very modest and we would like to be very modest, because we would have never gotten here without any of the community around Prometheus, without any of the support and attention that open source and monitoring is bringing to the table over the last few years.

Yeah, I think that if we could do anything to help other companies not have to build their own solutions or be able to contribute to like a single platform so we're not constantly reinventing the wheel and in a way that's like not going to break your budget, I would love for people to be able to help us all leverage one thing.

[00:57:45] JM: Rob Skillington, thank you for coming on Software Engineering Daily. It's been great talking.

[00:57:48] RS: Thanks, Jeffrey. It's been a fantastic experience. I really appreciate it.

[END OF INTERVIEW]

[00:57:55] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure

faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

Go to softwareengineeringdaily.com/HPE to learn about how HPE OneView can improve your infrastructure operations. HPE OneView has easy integrations with Terraform, Kubernetes, Docker and more than 30 other infrastructure management tools. HPE OneView was recently named as CRN's Enterprise Software Product of the Year. To learn more about how HPE OneView can help you simplify your hybrid operations, go to softwareengineeringdaily.com/HPE to learn more and support Software Engineering Daily.

Thanks to HPE for being a sponsor of Software Engineering Daily. We appreciate the support.

[END]