EPISODE 494

[INTRODUCTION]

[0:00:00.3] JM: You are requesting a car from a ride-sharing service such as Lyft. Your request hits the Lyft servers and begins trying to get you a car. It takes your geolocation and passes the geolocation to a service that finds cars that are nearby, and it puts all those cars into a list. The list of nearby cars is sent to another service, which sorts the list of cars by how close they are to you and how high their star rating is.

Finally, your car is selected and then it's sent back to your phone in a response from the server. In a microservices environment, multiple services often work together to accomplish a user task. In the example I just gave, one service took a geolocation and turned it into a list. Another service took that list and sorted it, and another service sent the actual response back to the user.

This is a common pattern; service A calls service B, which calls service C and so on. When one of those services fails along the way, how do you identify which one it was? When one of those services fails to deliver a response quickly, how do you know where that extra latency is coming from?

The solution is distributed tracing. To implement distributed tracing, each user level request gets a request identifier associated with it. When service A calls service B, it also hands off that unique request ID, so that the overall request can be traced as it passes through the distributed system. If that's confusing, don't worry we're going to explain it again during the show.

Ben Sigelman began working on distributing tracing when he was at Google and he authored the Dapper Paper. Dapper was implemented at Google to help debug some of the distributed systems problems faced by the engineers who work on Google infrastructure. A request that moves through several different services spends time processing on each of those services.

A distributed tracing request measures the time spent in each of those services. That time spent is called a span. A single request that has to hit 20 different services will have 20 spans

associated with it. Those spans get collected into a trace. A trace is like a tree of the different spans that a request has spent time on.
A trace can be evaluated to look at the latencies of each of those different services. If you're trying to improve the speed of a distributed systems infrastructure, distributed tracing can be very helpful for choosing where to focus your attention.

The published Google papers of 10 years ago, often turn out to be the companies of today. Some examples include MapReduce, which formed the basis of cloud era; Spanner which formed the basis of CockroachDB and Dremel which formed the basis of Dremio. We've covered these different papers from Google.

Today, a decade after he started thinking about distributed tracing Ben Sigelman is the CEO of LightStep, which is a company that provides distributed tracing and other monitoring technologies. LightStep's distributed tracing model still bears a resemblance to the same techniques described in the Dapper Paper, so I was eager to learn about the differences between the open source versions of distributed tracing, such as open Zipkin, and the enterprise providers such as LightStep.

One of the key features of the LightStep that we discussed is garbage collection actually. It's interesting to find out about how a distributed tracing system needs to do garbage collection. If you're doing distributed tracing, you could be collecting a lot of traces, because every single user request could theoretically give you a trace back and not all of these traces are useful, but some of them are very useful.

Maybe you only want to keep track of traces that are exceptional latent; really long latency requests. Maybe you want to keep a trace for the last five days, or maybe you just want to destroy them over time. You want to have different policies about how you retain these traces, because you're never looking at them, unless you have a specific reason to look at them.

The question of how to manage the storage footprint of these different traces was as interesting as the discussion of how to do distributed tracing itself. Beyond that the distributed tracing features of his product, Ben has a vision for how his company can provide other observability tools over time.

I spoke to Ben at CloudNative Con, KubeCon, and although this conversation does not really talk much about Kubernetes, the topic is undoubtedly of interest to people who are building Kubernetes systems. I hope you'll like it and all the other episodes about Kubernetes in the next couple weeks.

Thanks for listening and let's get on with the episode.

[SPONSOR MESSAGE]

[0:05:14.1] JM: Today's episode of Software Engineering Daily is sponsored by Datadog. With infrastructure monitoring, distributed tracing and now logging, Datadog provides end-to-end visibility into the health and performance of modern applications.

Datadog's distributed tracing and APM generates detailed flame graphs from real requests, enabling you to visualize how requests propagate through your distributed infrastructure. See which services or calls are generating errors or contributing to overall latency, so you can troubleshoot faster or identify opportunities for performance optimization.

Start monitoring your applications with a free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog and learn more, as well as get that free t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW]

[0:06:13.9] JM: Ben Sigelman is the CEO of LightStep. Ben, welcome to Software Engineering Daily.

[0:06:18.4] BS: Thank you. It's a pleasure to be here.

[0:06:20.6] JM: You were the author of the Dapper Paper, which was published in 2010. Dapper described the distributed tracing strategy of Google at the time. Describe the monitoring issues that Google was having in 2010 that were unsolved that you were looking at with Dapper.

[0:06:38.5] BS: Well, I'll give you a slightly more colorful answer in terms of where Dapper really came from. It's interesting. The paper was published in 2010, which is accurate. The truth is that Dapper was actually – the paper was originally written in 2006. I submitted it to conferences then. It was not accepted. Then I re-wrote it in 2008 and submitted to conferences and it was not accepted. I was like, "All right. Well, people aren't interested in this."

The conference proceedings, my understanding from the conversation is it's a very controversial paper and that everyone recognize two things. One, that it describes and that was useful to Google and also that it was of no scientific merit, the sensors of hypothesis that we are testing. It was just true. I mean, I'm an engineer. I'm not a scientist and we built Dapper to be useful and it described a useful system, but it wasn't intended to have any scientific contribution.

The funny thing about it, the only reason that paper ever saw the light of day at all was that someone at Google went to sight it for an actual academic paper that was testing hypothesis. They said, "Hey, Ben. Whatever happened to that paper?" I was like, "It never really made it." I was like, "I'll just store it online."

This is very haphazard thing that we did. It's interesting to me. I mean, I think there are good reasons on why academic conferences focus on asking scientifically novel questions, but there were actually things in the 70s and 80s that talked about the idea of tracing a transaction across some form of distributed system, whether it's a super computer or whatever.

The thing about Dapper, the reason the paper was interesting, to me it was a cool way to build a system at scale that actually worked. That's an engineering whitepaper, not an academic paper. We just threw it online, see what will happen. It turned out that it really is traffic cord and has been – I've gotten a lot of feedback about that paper. I'm of course not the only one who wrote it. There are other people who helped out with it.

With that amount of feedback about it being a nice way of describing how Google contended with a certain set of problems. Getting back to the history of it, those problems actually cropping up in 2004. The reason that Google pursued that technology is that they were going through the

process of splitting their own services into small pieces. They didn't call that microservices, because that word had not been invented yet, but that's absolutely what it was.

Rank and file, developers were having incredibly difficult time answering pretty basic questions. You had noticed that your – whatever service you were in charge of wasn't performing as expected and you had to explain why. That is a simple question to ask. A very simple question. Some transactions in your service are slow. Why? Just the first order why, not even like a full root-cause analysis, but like where is it slow even? They couldn't answer that question. Dapper was an effort to address that question. Help people who maintain systems both in firefighting situations and in more blue sky, we're going to make this faster over the next nine months initiatives.

We wanted people to be able to look at their system and figure out where the slowness was actually coming from. When you have a Google, a cache miss on Google web search would involve oftentimes many thousands of processes before it came back to the end-user a couple 100 milliseconds later. There's no way to go through thousands of processes to start calling people's desk phones and ask for them what happened. You have to have an automated record of how transactions propagate. That's what we were trying to accomplish with Dapper when we started that project.

[0:09:55.3] JM: To extreme ends of use cases, where you're talking about on one end you have situations where you need to debug something, because distributed trace is measuring the latency of different dependent services that are hitting one another. If latency is significant enough, then the service might as well not be working.

In that situation, you might as well be debugging. Then there's also the situation where you've got some remote call that works. It traces through 50 different services, but you'd like to improve it over time. Here we see distributed tracing both as a debugging tool, but also as a performance improvement tool.

[0:10:40.5] BS: Right. There is the firefighting, you're getting paged debugging. There's the performance analysis for that. A second, I'll drill into that a little bit deeper. What's so interesting

is that we had groups at Google who would spend a lot of time. I mean, talking like multiple engineer years on efforts to improve performance for their service.

They would do it actually. They'd make 20% improvements in the median latency, which is significant for services that had been around for a long time. It turned out that those improvements were off the critical path of a user request. Let's say that web search for instance would, just to use a common example, will talk both to the search corpus and also to a few search for – something that's in the news right now. You'll get articles at the top showing these headlines.

That means there's a request that goes to the main web corpus and also request the new service. Let's say that the news team makes their thing 20% faster, but it's always the fastest thing on the page. It doesn't matter. It literally doesn't matter. It doesn't affect that user's experience at all.

We had people who routinely were spending a lot of engineer time, improving things that actually were on the critical path. If you look at the data in the aggregate, you can actually determine well what is a critical path for web search and which service is going to most likely affect the critical path. Because if you have this big system, it's incredibly difficult as a human being to estimate just intuitively which service is actually going to have the greatest impact on the user-facing latency. Without tracing, you don't have the data to even answer that question, much less act on it.

It's a really interesting challenge actually. If you have a system of sufficient complexity to figure out where to spend your cycles on green field performance improvements, but you need a tool like that to answer those questions, I think.

[0:12:26.0] JM: Google did have some tools at the time though. What were the tools that they were using to do things that were – was there even anything like distributed tracing that they had deployed?

[0:12:39.3] BS: Not really. There were a variety of tools that people have always used for things like this. There's centralized logs. Those are always a thing. Metropolis centralized logs is that

it's difficult to pay for all of the logging data from all of the services in a central logging system. It sends too much data.

It's no different – I mean, tracing is really just a form of logging that has a little bit of extra structure. That's all it really is. The reason why it ends up getting into own a system is because you have to contend with a sampling problem as well. Tracing is a situation where you need to take all the flogging data and throw away almost all of it, and then the stuff you keep still has to be useful. Dapper solved that by only recording one out of every 10,000 requests. That was done totally randomly. LightStep, my company now has a very different approach to this problem, but there does need to be some kind of widowing with the data.

The trouble of centralized logging is it's literally too expensive. Even if you can make sense of the data on the other side, it's too expensive to centralize all these data. The other approach would be something like a metric system, or something like that where you can measure latency with a lot of precision. That's important, and Google certainly did that first with Borgmon and then with Monarch, which is another project I worked on actually after Dapper.

The trouble with that measurement is although you can record the latency of every system component, you can't explain it. There's a big difference between knowing that something is slow and then having a detailed explanation for why it's slow. The measurement is important and if you measure every component and it's in an egregious problem, you can often pick, you can often figure it out.

It's really hard in a general case to take performance measurements of slowness, even if they're across entire system and then pieced together the causality of it all. That's for tracing. It made things much, much more efficient and we've seen that in the industry at large in the last couple of years as well.

[0:14:25.0] JM: That's because there is not necessarily determinism for how your individual services are going to – how quickly they're going to respond to you, so you can't just say, "Yeah. Look, we're logging all of the request times from this service and we're just going to take the average." That's not how it works. You can't do that.

[0:14:45.5] BS: That's correct. Yeah.

[0:14:46.9] JM: Let's unpack some of the primitives of distributed tracing just in terms that will help explain what distributed tracing is and how people will use it. Could you just explain the terms span and trace?

[0:15:02.7] BS: Sure. That's a great question. The model that Dapper followed and which has been used by some, but not all tracing systems. There is also a system called X-Trace that came out of Berkeley, which has a different model. I don't mean to imply this is the only way to do it, but the way that Dapper did it, which I think has made a lot of sense practically, you have many services that are communicating with each other concurrently. It's totally common place in a trace to have the transaction taking place in many different services at the same time, because you'll send a request to many different back-ends concurrently and wait until the last one comes back before getting to end user, something like that.

Each one of those individual time space in each service is called a span. A span has a start time and an end time. It's local to a particular process and often even more granular than that, like a particular function call or something like that. The spans can have dimensions and tags on them, so you might tag a span with a user ID, or the path of an HTP call, or something like that. Whatever could be useful in providing context later when you're examining a trace.

Then the traces are – the spans are arranged into a trace via parent relationship, so you can think of the spans forming a tree and that tree is the trace itself. They unpack this, of course easier with a diagram, but let's say that you have a transaction to buy something and you have to through your bank system so you can imagine the transaction for the purchase is sent to the bank.

It first needs to call the balance checking service to see if you have the sufficient capital to do this. It will check that and that's its own span, and then it makes the call to actually change your account balance that maybe needs to go out to many different services to get them right into, like a consistent storage system, so you have three different service calls that go out simultaneously. You wait for all of them to finish. Then you maybe write into a cache, which is the final span, and then you're done and it returns the user.

At that point, you have five different spans to represent this trace and a user looking at this. It's like the Chrome debugger or something where you see all the timing. The only addition to that is you have this notion of structure, so you know which span cause, which other child spans, and that allows you to go through a trace even that has hundreds or thousands of components in it and figure out, "Well, I don't care about anything below that predictor spans. I'll just – I could, like a tree view just minimize everything there and focus on the parts that actually are relevant to your investigation."

That data model has been really effective for tracking interactive latency in real-time transactional applications, things where typical latency matters a lot. People want to trace systems that do say, non-latency sensitive workloads like Kafka queues, things that can take minutes, not seconds. Those systems sometimes have used other models, and if you look at academic literature and tracing that didn't get published, you'll see a lot of other ways of modeling this data.

The reason the span model has worked so well, I think that the developer, the cognitive overhead of understanding tracing and instrumenting for tracing is probably the greatest barrier to entry for that entire category of development tool.

[0:18:17.6] JM: People don't understand it.

[0:18:18.7] BS: Correct. Having a simple concept like a span, even though it's not always the right thing in all situations, it's probably 98% of the time and it makes it so much easier to do latency analysis if latency is built-in to every event that you record in your system. That is probably a worse of all compromise to use a span, even when there are some instances when you could probably suffice with just a single event that has a single timestamp instead of a pair of timestamps.

[SPONSOR MESSAGE]

[0:18:53.8] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes.

You can quickly provision clusters to be up and running in no time, while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked-in to any one vendor or resource. You can continue to work with the tools that you already know, so just helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

Check out the Azure Container Service at aka.ms/acs. That's aka.ms/acs. The link is in the show notes. Thank you to Azure Container Service for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:20:21.4] JM: Let's say I – just to generalize your example, I buy something on a website, let's say that hits the buy item service and we'll call that service A. You might have service A that makes concurrent calls to service B and C. Service B and C are going to need to result independently to get information back to A, then maybe service B calls services D, E and F and service C is calling services X, Y and Z. You have this totally disparate request paths and this is why you described it as a tree, because you just have these different trees of requests, and you could minimize the ones that you don't care about.

The spans aggregate into traces. Span is maybe the amount of time that it takes for a request to make its way through a given service in a chain of services. It's aggregating into a trace. This gets us to the question of sampling, because we may not want to trace every single request that

comes through our system ever. We want to sample. We want to have some of our requests be traced through. Is that correct? Is that the sampling?

[0:21:42.0] BS: Yes. I think, of course we want every trace to be sampled if we could. It's an issue of cost. I mean, Google's published number is that they're issuing 2 billion RPCs per second at Google. You're not going to record all these actually. It's not going to happen, even the 1 out of 10,000 there's a lot of data.

The challenge is to record the ones that people are going to want to look at and it's really hard to do that ahead of the event itself. The way that Dapper did it was to flip a coin and 1 out of 10,000 times it would record the transaction and the other 9,999 it would not. That's a very brute force approach to this.

If you're dealing with web search where they actually had several million queries per second on Google web search, that's fine. You still got plenty to work with. However, if you're using something like, let's say Google Checkout; people don't buy things off as they search for things.

The number of requests is small enough, that kind of sampling really crippled the utility of Dapper for projects like that. That sort of the way the ball bounces for Dapper, I think the sampling problem in my mind is the elephant in the room for tracing. It's not just about how many traces you can look at, but if you sample that data it's gone. You can't even analyze it and aggregate anymore. It's just gone. It's not written anywhere. If you want to do aggregate analysis, or if you want to do a very fine-grain predicate search over data, if it wasn't in that 1 of 10,000, that's the thing about it that's so problematic.

[0:23:13.3] JM: Interesting. We've given people a pretty good outline for what distributed tracing is. In 2010, the Google architecture looked a lot like what the popular architecture of applications today looks like outside of Google, or at KubeCon and that's basically the whole idea of Kubernetes is here is an externalized version of a Google architecture. You can do whatever you want with it.

We're seeing papers like Dapper that was published more than seven years ago. I don't know how long LightStep has a company, but now we're seeing – I mean, I guess for a while we've

seen plenty of other companies that were papers from Google turn into companies, so I guess we saw this with Cloudera, CockroachDB, a bunch of other stuff. You started LightStep and I want to talk about that.

When you were talking to people at KubeCon, are the challenges of logging and monitoring, or the challenges that people are having in Kubernetes today are they very similar to the challenges that people are having say seven years ago at Google?

[0:24:21.4] BS: That's a great question. There's certainly a lot of overlap. It's not something about Google having a more sophisticated system or anything like that. In many ways, I think the problems that people are resting with outside of Google are a lot harder.

Google have almost accidentally built a system that was incredibly consistent. I mean, they did have microservices, but they famously had this giant mono repo. I mean, they had this single repository of all of their production. In fact, they still do as far as I understand. That allowed them to be perfectly consistent about the way they did a lot of things.

I've never seen that kind of consistency any other organization that's at scale. It may exist, but I'm not aware of it. That consistency made a lot of things that are huge problems for enterprise in general and interesting problems actually. It just made them non-issues for Google, because they didn't have that how to reach an 80. It's a very homogenous regimen to the system in terms of the way they built stuff.

It's the same about what I see here is that people are contending with problems of scale in a way that they didn't use to have to. Because even if your business hasn't changed that much in terms of the number of transactions you're processing, if you've gone from having a handful of services around the monolith to having a hundred services and each service generates some amount of just baseline data, especially if you're talking about things that are per transaction. You've just multiplied your bill for anything that's cost-proportional to data volume by a couple orders of magnitude.

That's a huge thing. It's like a category breaking transformation, and that's I think why there is so much dismay from people who are trying to observe these systems, because even if the

tools work, they probably aren't affordable anymore. There is that issue and that is we definitely saw at Google.

The other fundamental thing is you see things like Prometheus and OpenTracing, these are incumbent about this. You see a trend towards instrumentation as code, like it used to be that if you're using "monitoring" in the early 2000s, you're probably just using something to like **[0:26:27.2]** to observe the actual physical infrastructure.

Maybe you got as far as looking at CPU and memory, but that was it. The idea of having metrics that are part of your code was a really new thing in industry at large, five-ish years ago. Now we take it for granted. That's a big deal. I think we see application level – level 7 application level things, and primitives and part and parcel of observing the system.

If you don't see it at the application layer, it's very difficult to understand what's happening. I see a trend in the larger ecosystem here, which resembles what we saw at Google and that direction as well where application code isn't complete until as unit test that's well known.

It also isn't complete until it has monitoring metrics built in, and I would argue if you're in the microservices deployment, it's not complete until you can convey to the world that you propagate trace information as well. Because if you don't, then you broke the trace for everyone beneath you and that doesn't work. I think that these things become first-class requirements for building software. That's a big shift, I think.

[0:27:30.2] JM: Who's responsibility is it to deploy those systems? Like, okay somebody sets up Kubernetes. Does that person also have to go through the setting up of Prometheus, distributed tracing, some like logging stuff? Because setting up all these stuff takes a lot of time, right?

[0:27:49.2] BS: Yeah. Absolutely. I think if you do it that way, it ends up being pretty long – it's a long road for that team if you make all that the responsibility of the infrastructure group. What I like to see and I've seen a lot of different models in my travels with LightStep, but the one I like the most is an organizational model where you have some kind of centralized team that sets best practices and –

[0:28:09.9] JM: Platform team.

[0:28:10.8] BS: Yeah, something like that. Or sometimes they're called the insights team if they're focused on observability. You have people who set best practices and actually set requirements for how services must be deployed. It becomes a checklist that I'm sure is frustrating for people and I respect that frustration.

It's also much better for the organization to have it that way and to make sure that it's what I said. It's like you can't deploy into production unless you have your monitoring checked off. I think that model is more sensible.

It's too difficult, even if you ignore the time cost of doing the work and centralizing that. It's very difficult for a central infrastructure team, because of what I was saying about instrumentation being part of the code. It's very hard for them to know what metrics are going to matter to a group, so all you can really do is measure top-level things like HTP calls or something, which is fine, but it's often not what you actually care about.

I think it's the responsibility of the person building the future, or writing the code to decide how to monitor this. The best thing I've seen is when there's a group that decides what it means to check the box, but then the individual teams of dev ops or devs, they have to actually do the box-checking themselves.

[0:29:17.9] JM: We've articulated some differences between building a distributed tracing system for Google and building distributed tracing system for the broader populous, namely Google has this mono repo where deployments are a little more restricted by that, or they're a little more controlled, or more consistent, or maybe the services instances are more consistent.

In any case, there is a difference there in how people are deploying outside of Google versus inside of Google, so I think this gets us to a segway into building LightStep. What are the differences between building a distributed tracing framework for internal consumers of Google and building of distributed tracing framework that can be used by the broader packet list?

[0:30:06.8] BS: Yeah, it's a great question. It's interesting. I mean, LightStep is a company – we don't position ourselves as a distributed tracing solution. I use that terminology at KubeCon, because that's what the people want or whatever and that's what people are calling this.

I think as an industry, we need to do a much better job talking about what our use cases and value proposition and last about what the particular technology choices are. I actually think distributed tracing of course is really important and it's accurate to say that LightStep has a distributed tracing system within it.

However, we really think about use cases and workflows and that use cases that we think about have to do with measuring symptoms that are crucial for a business about performance, but across the system. Not just application handlers, or microservices but even individual customers, individual users, individual releases were very flexible about the symptoms that we measure. Then we're also flexible about how people do root cause analysis for anomalies in those symptoms, and the root cause analysis pieces where the tracing comes in.

We don't actually internally talk that much about tracing. We talk about workflows and use cases. I think that as a collective of open source projects, the world would probably be a little bit better if we focus more on use cases and workflows for open source projects as well.

Tracing is part of a monitoring solution. It's not the monitoring solution. It never will be on its own. You have to be looking at the right traces with the right context to drive value from them. That's more than just a distributed tracing problem. That's like a stream processing problem and analytics problem and a UI problem and so on and so forth. That's really how we think of it.

Going back to your question about it differs from things with Google, I think to a certain extent because Google is just such a nerdy place and I'm such a nerd, I was probably a little more focused on the implementation of the tracing system per se than on these workflows when I was at Google. You also don't have the economic pressures on you that you do if you're a company to deliver value.

One thing I love about being a vendor is that you can tell if you're delivering a value, because people buy your stuff or anything like that. At Google, it's always actually really frustrating and

that there's never any budget for anything. People just built stuff and if is useful, great. But you wouldn't necessarily know about it.

There's nothing like knowing that someone is paying for something and renewing to be like, this is actually valuable to them. I like the economics of being a vendor, because it forces you to think about delivering value in a very real way. Not about swindling people about giving them a really good deal. It's like they're getting something way more valuable than what they're paying. To me, that actually really changes the orientation of the conversation internally about how we design product.

Getting back to your question, that orientational thing is actually a big deal. At Google, we are mainly concerned with getting these traces in a place where people could see them and contending with scale, and that was sort of it. With LightStep, we're much more concerned about use cases and workflows than we are strictly about just building an awesome tracing system.

The awesome tracing system followed from the workflows, I think. There are a lot of differences between what you've done at LightStep and what you did with Dapper. They're actually totally different approaches. For that reason actually, I think that we thought about use cases and it led to a completely different approach to building the system itself.

[0:33:17.4] JM: As a vendor thinking about workflows, that's like a top-down development approach, because you're thinking, "Okay, we've got customers who are going to deploy in this certain way and they're going to run their infrastructure in that certain way. What is the best monitoring tool that we can build for them. What is the best workflow that we can give them. Am I understanding it correctly?

[0:33:40.6] BS: That is correct, and it's often something where the workflow already exists. If the customer is using Refana and they're really happy with it, that's great. We have no intention of displacing Refana. It's a great piece of software. What we do have an intention of doing is making sure that LightStep's data shows up in Refana as part of your dashboard, because that's the workflow you're already accustomed to.

I was just thinking about this the other day. I use VI. I've used it since 1999. Is it the best editor for me? Almost certainly not. But you know what? I'm never changing. It's never going to – if I was going to change, it would've already happened. It's not going to change. I think that's how most of us are for developing software, we get into a habit with the tool that we're accustomed to and asking a developer to change their tool chain is a big thing. Even if it's better, it's a big thing.

I'd rather see that LightStep's data is really valuable and is integrated into the tools and workflows people are already familiar with. That's the way that we're trying to build a product. Of course, we do have a frontend with dashboards and so and so forth and you can certainly use that if you want to. But we're pretty agnostic about that. We're not trying to be – we don't attempt to be the one tool to rule them all. I think we see the data as being quite valuable, and especially when incorporated appropriately into our workflow is even more valuable.

We don't see LightStep as being the thing where you throw all your other bookmarks. You jus bookmark LightStep's application and then say you're done. I think that's philosophically something that we feel pretty strongly about, and I think it served our customers pretty well as well.

[SPONSOR MESSAGE]

[0:35:13.5] JM: If you are building a product for software engineers, or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an e-mail jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers, because I talk to them all the time. I hear from CTOs, CEOs, Directors of engineering who listen to the show regularly. I also hear about many newer, hungry software engineers who are looking to level up quickly and prove themselves.

To find out more about sponsoring the show, you can send me an e-mail or tell your marketing director to send me an e-mail jeff@softwareengineeringdaily.com. If you're a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company.

Send me an e-mail at jeff@softwareengineeringdaily.com. Thank you.

[INTERVIEW CONTINUED]

[0:36:41.5] JM: You're saying when you're going to a customer like Lyft, for example, I think Lyft is a customer of yours – I did a show with a company called Wavefront a while ago.

[0:36:51.9] BS: Yeah. Sure. I know them well. Yeah.

[0:36:53.2] JM: Yeah. That's another analytics company, like health monitoring, stuff, but you're saying that you could deliver, even perhaps to a team that deploys a product like Wavefront. You could say, "Oh, we can actually give you additional insights, additional – uncover additional problems."

[0:37:12.2] BS: Absolutel. Yeah. Let me say a little bit more about LightStep. We talked earlier about sampling in Dapper and how Dapper would throw out most of their requested 99.99% of it, before it would record anything.

What we did with LightStep is quite different. We actually do record all of the data. We have a 100% of the data, but not for very long. For order of minutes, we have a 100% of the data stored in these highly optimized certificate buffers that run close to the application, and the customer's own DPC or private cloud, or physical bare metal data center, whatever it is, they run component of our system there, because there's so much data coming in that if you sent to over the wide area network it would cost you millions of dollars. There's a ton of data going into that pool.

The pool is communicating continuously and bi-directionally with the LightStep SAS. That design allows us to do some pretty cool things, where the pool has a very short memory, but as long as we can detect that something is slow within that memory. If you're talking about user interactive latencies so things would take a couple of seconds or less, we have more than enough time to detect that something is slow.

If there's the P99.99 trace in terms of latency, we will always record it. Always. If it's a median latency, we'll record some of them, but not all of them. If you send us – like Lyft sends us a million spans a second, so that's fine. We don't record every one of those forever. We'll certainly record them temporarily in our circuit buffer. We can go back after the fact and say, "Well, this one is an anomaly. This one had an error. This one is slow." This one matched a very specific predicate and this is getting back to what I think is so cool about this with the graph on it and everything like that.

Twilio is another customer and they LightStep to –

[0:38:54.6] JM: They probably also use that.

[0:38:56.7] BS: Probably and vice versa actually.
[0:38:58.2] JM: Yeah, vice versa. Hilarious.

[0:38:59.5] BS: That's actually a foolish case study. I will say it's totally going to – all the contracts who never do this, but it's funny because Twilio and Lyft are both customers. Then Lyft is a customer of Twilio. We actually see traces from Lyft going to Twilio and then we see traces from Twilio coming from Lyft. We could connect the dots all the way through. It's a really interesting thing for us. Of course, we don't because we're not allowed to, but it's academically interesting to me how that would all work.

Anyway, getting back to the point. Twilio, well this is a good example. I'm sure Lyft is an important account at Twilio and for their important accounts, they have LightStep measuring the performance of individual accounts. When there's an anomaly with an individual account, they can remediate that specific anomaly. They're able to focus on something that's granular and specific customer.

We also have a VP of engineering that's a buyer for our product at another company, I won't say which. He uses LightStep to track the performance of his board members, because he goes into board meetings and gets chewed out for performance. He can say, "Well, there's only five of them. Let's have contract that." He has a save search in LightStep to track the performance of these five human beings and his product, and then he can go back and see why they were slow. He could say, "Well, I know exactly why that request is slow," even though it's just a little sliver of their entire production traffic."

By recording everything, we can be incredibly granular about what we actually measure. We can also go back in time and assemble traces really for any arbitrary criteria. That's a really important, important thing to understand and I think separates LightStep from a technology standpoint, from anything that I'm aware of. That's where most of our effort has gone is to making that work and that's why the customers are excited about the product.

[0:40:38.0] JM: Okay. This system that you described, the open tracing system you described is basically a system that gathers all of the traces. I mean, this gets at the sampling problem we discussed earlier. You don't want to gather and save every single trace through every single request, because that is a huge resource consumption problem. What you've essentially done, if I understand what you said correctly is built a garbage collector for distributed tracing.
[0:41:07.3] BS: That's a great way of putting it. I've never thought of it that way. I think I'll use that. Yeah. That's very accurate. When we do record a trace, just to be clear, we'd record it forever, so there's no time horizon on those recorded traces. If we say a trace is interesting, we keep it forever and it has a permalink and so on and so forth.

The ephemerality is just for the data before we've made decisions about importance. That means that the customers in the UI if they see a link to a trace, that trace is always going to be there. It's only the data that gets dropped that they never see that they won't be able to access later. That's an interesting design tradeoff that we've made, but I think informs most of our capabilities. That's the part that's deviated entirely from what Dapper did.

[0:41:48.5] JM: There's nobody that's built an open source version of a distributed tracing garbage collector thing?

[0:41:54.0] BS: I'm not aware of one. It's a difficult thing to do technically, and when they started the company before we took on any money we were testing that hypothesis that it was literally possible. I think we didn't know if it would work or not at production scale. That was the biggest technical risk we took with the company. I mean, most of our engineering time has gone into making that piece work. That's been a very difficult thing to build.

[0:42:16.7] JM: Can you talk some about that?

[0:42:18.3] BS: Sure. I mean, it's difficult to talk about in a lot of detail just because it's just talking and it's one of those things that demands a whiteboard.

[0:42:26.0] JM: I imagine. Okay. Just to help people understand, if you're Lyft and you rolled your own tracing system, you are gathering tons of spans. You're aggregating the spans into traces and you're doing some kind of sampling, because you can't collect all of these yourself. If you wanted to build some garbage collection system, you would have to decide which of these I'm throwing away. When I'm throwing them away, you're describing a system LightStep that gathers all the spans gathers them into the appropriate traces, figures out which traces are useful and keeps those, stores them and then throws the rest of them out.

In order to do that, you built some in-memory system that takes in traces and figures out which of those are interesting. Maybe can you talk a little bit about the – we did another show about distributed tracing. Is there anything that's different about the distributed tracing model itself, or is the secret sauce really in this garbage collection log rolling thing?

[0:43:32.9] BS: There are some differences in the tracing model. Yeah, it actually goes back to the 100% of the data again. I've given demos about this. I did a talk at Monitor ML last May where I showed this onscreen, so if people want to look that up, you can. But the thing that's interesting about latency is that if you look at latency – variable latency, like where you're operating the system and then suddenly latency spikes, but you – it's not like it's a new version of the code. It just spiked.

I'm not going to say all the time, because that's irresponsible. I think in the presentation I did say all the time, but I regret saying that. I will say a large portion of the time, if you have that symptom the issue is actually contention for some shared resource. Maybe that there's a database table that's too hot. Maybe there's a mutex somewhere that has too many people waiting on it. It could be contention for a network. It can be contention for any number of things.

Contention is actually really the issue. You can do a latency analysis with distributed tracing. It's great. It's better than nothing. You'll get to the point, well you'll see, well this thing should've taken one second. It took five seconds. Four and a half of those seconds, we're waiting on this one span.

Then the immediate question is why did that span take so long? That is where Dapper stops and every other tracing system I am aware of stops at that point. What we have that opportunity to do with LightStep is we can actually say, "Well, that was contending on a database table. Why don't we just look at every other transaction in the entire world that was also contending on that database table right now and then figure out where it came from?"

We can do that analysis. It's not difficult to do. I mean, it does require some work, but we have all the data and the small index appropriately to make that work. That kind of thing is really – that's like a holy grail for me anyway as someone who has worked in this area for over a decade to be able to do that analysis.

It does require additional tagging of some resources, like there needs to be some identifier for each resource that you can tend on, that whether it's a table name, or thinking the idea of a mutex or whatever. If you have that identifier, you can in a very principled way, you can figure out where the load is coming from. That is like a big deal in my mind, and it goes way beyond looking at an individual transaction. Now we're talking about interference between transactions. That's actually where it mostly in the seat comes from. That's different I think.

Looking at individual transaction, yeah. I mean, there are things about it that are a little bit better and we do a nice job as a critical path analysis and things like that, but nothing profound. It's only when you start looking at interference effects and I think we're doing something that's really different.

[0:46:02.3] JM: Okay. Well, talking more about the infrastructure, I guess this is your SAS, where you're gathering the tracing and deciding what to throw out and deciding what to keep. Can you talk a little bit about the – just giving a depiction of your infrastructure, what are you running on, what does it take to build this garbage collection system. What are some of the components that go into it?

[0:46:23.9] BS: Sure. That's a good question too. The backend is really a 100% go. I think there is nothing that's not go actually except for the react frontend. The challenge for us all along has been to make intelligent decisions about what we decide to build ourselves and what we can just use something off the shelf.

We we're using Cassandra for a while and we had a near miss with an outage. We had our own meta-monitoring that was monitoring our main instance, and the meta-monitoring broke in a profound way with data corruption and Cassandra. It's fine for the operate Cassandra according to the user manual, but we got a little freaked out about data corruption in our data store.

We moved away from Cassandra and have been using Google cloud spanner actually. Actually very happy with it. I have a enormous amount of respect to people who built that system. It's been for the workload that we're sending, it's a very appropriate thing for us to use and has pretty amazing availability because of the time –

[0:47:22.5] JM: Why is that?
[0:47:23.1] BS: Why is the availability so good?

[0:47:24.6] JM: This is a good question that I wish I – I've not been able to ask the spanner people yet. I've done some shows about CockroachDB, which I think is similar to spanner.

[0:47:33.4] BS: Very similar.

[0:47:35.4] JM: What are the workloads that you want to run on something like CockroachDB or spanner?

[0:47:41.2] BS: We are lazy like everyone, and we also care about uptime a lot. The idea that you can get multi-region availability and consistency without any effort whatsoever is incredibly appealing to me. That's one of the checkboxes.

Then the other one is that we don't – for that particular workload, what we're using it for, we don't care deeply about throughput or right latency. It's best effort latency on the right side. It doesn't matter if it takes half a second, instead of a 100 milliseconds to write to that data store because of the way we're using it. It's not on the user path.

That deals with the – the biggest drawback to the approach that – not Coackroach, but that spanner has taken in that – there is always a penalty for anything that's beneficial. That's just straight out to everywhere, and the penalty for all the consistency and the global distribution is additional right latency and some issues with throughput. This workload isn't that intense in that regard. It's something that it's certainly it's too big to fit in one machine, or we would just put up one machine. It's a small enough workload that we're not stressed out about throughput or latency, especially on the right path.

Then on the read path availability is amazing. I mean, I would be shocked if that system ever was the reason why we had an outage. That's great. We don't have to think about it. We can be lazy. That's the workload for me. I mean, I have a huge amount of respect for the Cockroach team as well, and they've built an amazing piece of technology. I think if the timing had been different, we might be running that ourselves.

Anyway, going back to your question around how we built the system. It's all go. We use a lot of GRPC internally and it's one of those things where there are pieces of the system that we've used off the shelf things, like the spanner we're using just a normal SQL database for the boring stuff, just user accounts and things like that.

All of the tracing stuff, all this just takes all of the sampling, all of the commands that are sent back to our collection system from the SAS, all that stuff is completely utterly accustomed for better or worse, and which makes it difficult to describe. I mean, I think it's all for a reason, but we were unable to find anything that did this. Or as a startup, we're always looking to cut

corners in time to market, but that wasn't a corner we could cut without messing up the performance characteristics of the system itself. Those are critical to the product.

[0:50:05.4] JM: One component I can think of that you would need that maybe is hard to build is like a highly available consistent replicated in-memory store, because you need to keep all those traces that are coming in memory somehow and you probably want to have some amount of replication, or –

[0:50:27.2] BS: That's a good question. We keep statistics and the traces themselves. The statistics –

[0:50:32.8] JM: The traces themselves, or everything in that.

[0:50:34.5] BS: Yeah, exactly. The statistics, we ship out literally every second. As they're shipped out, they're stored durably in a way that – unless something really catastrophic happened, we would never lose that data. The trace data is not SAS-compliant. No one is using this to count ad clicks to build our customers.

This is operational data. I would say we have probably four nines of reliability on the collectors, but it's not meant to be perfect. That's the reason why it works. If we wanted it to be more durable, we would be writing to disk and then it would never scale. It's an in-memory only circuited buffer. We certainly do things with load chatting to make sure that if we're bringing these collectors up or down, they don't lose data in normal operation.

If the power went out, like wholesale power went out, you'd lose probably a half a second to a second statistics and you would lose the trace data for the last couple of minutes. The product will reflect that. You'd see a gap. It wouldn't be in complete traces. I'm completely comfortable with that. That tradeoff is what allows the product to work, and I think that the trouble with using off the shelf stuff is that they have durability requirements that are general purpose. This is not a general purpose situation. This is operational data.

It's actually okay to have, as long as it's an uncommon occurrence, I think it's okay for there to be a certain amount of data loss in the event of a catastrophic like system failure where you lose

power or something like that. Trace data isn't actually honestly going to be that interesting that case anyway. It's just going to say that everything broke and if it's a power issue, and that's the thing that would take them down.

[0:52:06.5] JM: Yeah. I did want to ask you some about building a company in this space, because I'm walking around in the expo hall downstairs and just getting a feel for what's going on in the conference at KubeCon. It feels like there's a very different space than any prior analog. You look at the different periods of time we've had. I mean, the software industry is fairly young, but you think about maybe the big trends and going to Strata Conference five years ago or something, yeah there is a ton of vendors there and that was a new world, or maybe six or seven years ago, something like that.

This feels like a new set of vendors. It feels like a new period of time for building a company. Is there any unique challenge that stands out about building a company in the Kubernetes, cloud-native provider infrastructure space?

[0:53:05.4] BS: Yes. I mean, we're having a great time for what it's worth. I've never been this happy professionally as I am now. That said, I mean one of the things that is a challenge for us is to make sure people realize that – how do you say this? There's a bit of an impostor syndrome around the Kubernetes world, where I think a lot of companies like, "Well, we're not really doing it right. We're only partially on it." That's totally normal. No one is a 100%.

The biggest thing that we talk to prospects is like you don't need to be fully on microservices to benefit from technology like the stuff that we've built at LightStep. In fact, none of our customers, including the ones that feel very bleeding edge, like Lyft for instance, none of them have gotten rid of their model.

If it's still there, it's smaller, it's less important, but it's still there and I don't think it's going away anytime soon. That's normal. It's completely normal. I think the biggest message for people is you don't need to be a 100% transition. We have customers who run microservices alongside JVM stuff from the 2000s, alongside legacy mainframe, all of which somehow get their data into LightStep to tell a coherent story about transactions across all those systems.

It's fine. It's completely fine to integrate multiple generations of technology, public cloud, private cloud, bare metal, whatever. I think that the biggest thing is for people to realize that although the key notes here might indicate otherwise, nobody has all the way over onto this stuff, except maybe Google, Facebook or something. I mean, it's a very uncommon thing to see a company that's even 80% microservices at this point.

[0:54:31.5] JM: Last question. How do you decide pricing in a product like this?

[0:54:35.8] BS: That's a very good question. Something we thought a lot about. It's mostly how we don't decide pricing. Because of the data volume issues, one the thing that's so cool about LightStep is that we can absorb a lot of data. We actually have the customers provision these collectors that that's the name of the thing that absorbs all the data.

They provision in themselves and their own resources, which gives them complete control over the equation between how much data do you want to send, how many collectors do you want to run and how much recall do you want to have, which is to say how long do you want that circuited buffer to be. Because if you make it an hour, then you have an hour with perfect fidelity. If you make it 10 minutes, you have 10 minutes that's perfect fidelity. That's up to the customer to decide how to provision that.

It ends up being a small fraction of the TCO for the product, but they have control over all those knobs and we don't charge a dime for that. Similarly, we don't charge for seats, we don't charge for servers, we don't charge for containers, all that stuff is actually not the value of the product. It's something that's sometimes correlate with value, but not necessarily.

We do charge on is the analytical features in the product and those are literally the value. Every time you engage with LightStep in the commercial context, I mean all of the customers that we work with now are enterprise companies, so all these contracts involve a lot of scoping and customer-specific negotiations and so on.

The way that we think about pricing, there's a certain amount that we need to ask just to support a customer with – we have a very good support team with 24/7 escalation so on and so forth. There is that aspect of things. But for the services themselves, we think about pricing in terms of

the analytical value we deliver in the units that we price along, have to do with the analytical value, not with the scale of the data you send to us with a number of host or anything like that, which don't actually scale value. This aligns our incentives with our customers, and I think it's actually been quite beneficial for both parties in terms of making everyone feel good about the transaction itself.

[0:56:26.8] JM: All right, Ben. Well, thanks for coming on Software Engineering Daily.

[0:56:28.8] BS: It's a pleasure. Thanks.

[0:56:29.6] JM: Okay. Thank you.

[END OF INTERVIEW]

[0:56:32.8] JM: Simplify continuous delivery with GoCD, the on-premise, open-source, continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment workflows using pipelines and visualize them end to end with the value stream map. You get complete visibility into and control over your company's deployments.

At gocd.org/sedaily, find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent predictable deliveries. Visit gocd.org/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery are available. Thanks to GoCD for being a continued sponsor of Software Engineering Daily.

[END]