# EPISODE 1451

[INTRODUCTION]

**[00:00:00] JM:** Observability consists of metrics, logs and traces. Lightstep is a company that builds distributed tracing infrastructure, which requires them to store and serve high volumes of trace data. There are numerous architectural challenges that come with managing this data. And Ben Siegelman and Alex Kehlenbeck join the show to discuss the implementation of Lightstep.

[INTERVIEW]

**[00:00:21] JM:** Guys, welcome to the show.

**[00:00:23] BS:** Thanks for having us.

**[00:00:24] AK:** It's good to be here.

**[00:00:26] JM:** You started Lightstep around the time that container orchestration was becoming popular, monitoring, and logging, and tracing were becoming more prominent, and the observability stack was changing. And one of the trends around then was Google infrastructure for everyone. And since you have started Lightstep, the industry has evolved. And I'm wondering how Lightstep has evolved relative to the Google version of infrastructure. When you think about what infrastructure experience you guys had at Google, and how that compares to the Lightstep version of observability today, how do they compare?

**[00:01:11] BS:** That's a good question. So just by way of introduction, I'm Ben Siegelman, and I was one of the cofounders of Lightstep. And I'm joined by Alex. Alex, do want to briefly introduce yourself and your background?

**[00:01:24] AK:** I'm an engineer, a software engineer. And I've worked for Ben, with Ben, for many years. First at Google –

**[00:01:31] BS:** Alex and I met in 1947.

**[00:01:34] AK:** It feels like it.

**[00:01:34] BS:** No. I mean, I think we actually met and started working together in 2005?

**[00:01:41] AK:** I think we met in 2006. I'm pretty sure. You gave a talk about Dapper **[inaudible 00:01:45]**. That's how we met.

**[00:01:47] BS:** That's right. That's right. Yeah, Alex and I have been working together for a long time. And Alex is the brains operation for most of the heavy-duty observability stuff we did at Google. And we worked together on both Dapper, which was Google's tracing system; and Monarch, which actually Alex came up with a name for Monarch. And it's a monitoring architecture. That was the joke. But it's Google's central kind of time series, collection, storage, query analysis solution, kind of like a metric solution today.

So we have spent a lot of time seeing things like Google. And then at Lightstep, it's quite a bit different actually and for a variety of reasons. I think the biggest – I don't know what you think, Alex. But at Google, things were so big that you had to tradeoff scale and feature set. And so, there's this totally mistaken idea that people should try to emulate what Google did. And a lot of the time, that's a terrible idea. Because if you're – I don't know what the public – I think the public number that I've remember that Google's doing 5 billion RPCs per second or something like that. I think the actual number is quite a bit larger than that. But that's a ridiculous scale. And if you're going to operate at that scale, you have to throw out like almost every interesting feature you might want to build. And that goes for observability and monitoring, but also goes for many other things, too.

So the thing that's a bit sad to me is that there's been a tendency to emulate things that were done at Google and Lightstep, for what it's worth, in many ways was a reaction to the work that we did at Google, or a version trying to attack the same problem but with technology that was better suited to the scale of most enterprise software organizations, which are actually quite a bit smaller than Google in terms of that top line throughput number. Alex, I don't know what you think. But that's like my super zoomed-out quick take on things.

**[00:03:36] AK:** Yeah. And I think that's totally true. I would just add also that this is true of Google. I think it's true inside the ecosystem of any other large company. The clients, the things you're monitoring are just much more homogenous and well-behaved, right? We controlled a lot of the client instrumentation software that was on the client side. And so we could often trade off functionality there against things that we either did or didn't have to build in the monitoring system. And there's a lot less flexibility when you're dealing with sort of a third party and trying to monitor their stuff. You then have to do a lot more work to meet them where they are typically.

And then I would say on the flip side, on the product side, I hope we talk more about this in a few minutes. But I think that our thinking, especially Ben’s thinking, has evolved a lot since back then. We were very much – I'm going to say a dirty phrase, right? But the three pillars were things that like we believed in and everybody sort of believed in. This was back in 2009, 2010, and even for quite a bit longer. And I think the industry as a whole, but also our own thinking has come a long way.

**[00:04:34] JM:** Those three pillars being metrics, logs and traces?

**[00:04:37] AK:** Yeah, exactly. Yeah.

**[00:04:39] BS:** Yeah, totally. I mean, I think what happened at Google is quite similar to what I've seen happening in many organizations now as they move to kub and things like that, where you obviously have a logging system. You've had that since the dawn of time. You probably have somewhat robust metrics system to do squiggly lines and charts and things like that, infrastructure all the way up the application stack. And then you get a distributed system. And you already have like a ton of investment and debt in your metrics and logging solutions. And so you just add tracing. It’s like its separate thing. And then you end up basically with like three UIs that might be connected by hyperlinks and tabs, but you ended up with these three UIs.

And I think I remember when we built monarch, that was pretty greenfield, actually. Actually, it might be interesting, Jeff, if you think your listeners would be interested to hear a little bit about Monarch and what preceded it, which is this thing called Borgmon, which is Prometheus is modeled after very directly. And we could talk about that. But part of the pitch for Monarch was trying to do a little bit of very light unification through exemplars. So the idea was that you're going to be able to do monitoring via Monarch of latency. And then for latency outliers, you would have example trace links that you could go and look at. And we built that in the initial prototype for Monarch that we built before we invested an enormous amount of time and energy into that project had that functionality as kind of a spike.

So there was like some very loose integration, and I would consider that be very loose integration. But the actual value, particularly of the tracing data, is not the traces themselves. It's all the information encoded in the traces, especially in the aggregate. And at Google, we never took advantage of that in any of the real-time analysis stuff and where people are actually solving problems and responding to incidents, at least not before I left. Alex left after I did. But I don't think – To the best of my knowledge, that never happened.

And at Lightstep, we've dumped a ton of energy into that, because I think that's really where the space is headed. If you have distributed tracing, it's more that data is – It's like the underpinnings for a lot of analytical features that don't actually show you traces in the UI. And I think that's something that we've spent a lot of time kind of laboring over and building functionality around.

**[00:06:50] JM:** So can you tell me about the data path for collection and storage of metrics, logs and traces?

**[00:07:03] BS:** You mean at Google or in the sort of your average –

**[00:07:08] JM:** Well, I guess, in a broader sense. Like, give me an outline of where metrics, logs and traces get collected and the path – Maybe, I guess, in broad brushstrokes, the path it takes to get stored and queried. And then we can talk about how you do it at Lightstep today.

**[00:07:29] BS:** Okay. It's a tough one, because there's so many different ways that it's done. I mean, Alex, do you have a quick take on that? I can –

**[00:07:36] AK:** I mean, not a quick one. I think there's definitely not one size fits all. And it's another work at OpenTelemetry, which is an open source project that then was involved in founding a sort of predecessor of it, and Lightstep is involved. But there are many other companies involved. It's kind of an open source instrumentation layer that aims to kind of unify a lot of the question you just asked, “How does the data get out of the process, out of the application, and to somewhere else, to a vendor?” is aiming to unify that. But people are all over the place right now, right?

So the kind of simplest version is you have a sufficiently intelligent or fancy instrumentation library, and your process itself is making direct RPC or HTTP calls out to just push the data directly to a vendor. But people have all sorts of things that they layer on top of that or in between the application and the vendor, either for security reasons, or sometimes to decorate the data with additional information, right? Sometimes your process doesn't actually know itself all of the tags or the attributes that you want attached to the data. And so you put some intermediary in the middle that does know that metadata information, and it attaches it as the data goes out. Sometimes you ask the vendor to do that on your behalf after that your data has already reached the vendor. That wasn't a very good answer. But the real answer is that it's all over the place. People are doing all sorts of things to get the data out.

**[00:08:56] BS:** What I've noticed – And, actually, I was actually just talking to some folks from some of the Fluent Bit maintainers yesterday separately, and we were talking about some of the logging data. And for logging in particular, the data is used both for what I would call “observability use cases”. So like instant response, ad hoc analysis, that kind of stuff. But it's also often used for security, or for even like bin counting, and like we want to use the logging data to figure out how much money we made or something along those lines. In which case, it's extremely important they don't drop it.

And in those cases, there's a desire to have the logging – The logging infrastructure actually has to run like in the same VM as the process, so that if there's a network issue, it can buffer and catch up and things like that.

To my knowledge, I've never seen that. That need has not existed, to my knowledge, for instance, tracing data, and very rarely for metrics data. So I think there's a little bit of tolerance to data loss. I mean, not huge, of course, but a small tolerance data loss for metrics and traces, which from much of the logging is not like as acceptable, I think, because of those business reasons.

Ironically, with Dapper, we ended up, just for expediency, reusing the logging path that, again, was also used for things that were pretty vital at Google and were written to local disk. And that was a huge mistake. In the paper that we wrote about Dapper, you can see like where the system hits some scaling limits in the sense that, like Dapper – Full-bore, 100% sampling Dapper actually has interference effects and will affect the application you're observing. Like, it'll create latency and throughput issues. And that was largely because we reuse the logging path that writes local disk. And the second, you're like taking a ton of tracing data and making sys calls about that, you start interfering with applications that are also trying to make sys calls, especially to local disks that at the time are actually pretty slow.

So the long and short of it is that the logging path for business reasons, I think, ends up being more expensive than it needs to be for observability use cases. This is one of the many reasons that I frankly, think most logging should probably die, at least by volume. It's just like not the right tool for the job.

The tracing data is a little different than logging and metrics. With logging and metrics data, you can think of as a true pipeline. I mean, you can decide where you want to send things over the network and where you don't. But like there's a pipeline until you get to some central data store. With the tracing data, it's a lot worse, because there's also this assembly stage that you can do gradually or all at once. But you have to take the trace, which, by definition, is distributed, and you have to take it and assemble it somehow. If you assemble all the traces in a large application, well, suffice to say, is super, super expensive to the point that you aren't going to do it. So you have to be selective with traces you decide to assemble. And there's a whole nasty, nasty, complex set of optimizations that you have to consider around that, and especially if you want to be smart about which traces you assemble. But the full end-to-end lifecycle for the trace data is not so linear, where you have to assemble enough to decide which traces are interesting, then go back in time and actually do the assembly for those interesting traces if you want to do it well.

And we started – With Dapper, we did not think that through, frankly. But Lightstep, we did. And it has massive architectural implications to the whole collection pipeline. And it's an area where I still think there's a lot of catching up to do to what's possible. But the tracing one is worse than the others because the data structure itself is distributed.

**[00:12:24] JM:** And as far as the storage system for Lightstep, for metrics, logs, and traces, are you using – I guess I'm a little unclear. So do you use any off-the-shelf databases to serve it? Or can you differentiate between like the storage and the serving layer? And if those are different?

**[00:12:49] BS:** Alex's, it’s all you.

**[00:12:50] AK:** So we have a bespoke homegrown thing that we, my team, wrote, and it’s model than a bunch of lessons that we learned from the very big systems that we learned at Google, but at this point is diverged quite a bit in the details. But in sort of broad strokes, it's an LSM, a log-structured merge tree-based thing. It's pretty efficient for high volumes of writes. We index a bunch of stuff, all the data as it comes in using the dimensions that we know we’re going to care about at query time. And then the query serving, the query serving, and to some extent, also the kind of arithmetic that we like to do on especially the metric data, the aggregations that, the rate computations, and so forth. The serving and the arithmetic is also built into that same process within the same database. And the thing is distributed. This is sharded by customer, for large customers across many machines. And you have this sort of standard serving tree roll up from lots of leaf serving machines, all the way up to a single route to get your final answer. So that part of the architecture is pretty traditional, that in all of the details of the software we have sort of a homegrown specialized thing.

**[00:13:55] BS:** Alex, what was the – I mean, I sort of know the answer. But what's the problem with using like an off-the-shelf, horizontally scalable database for this sort of data?

**[00:14:04] AK:** Yeah. So at the risk of speaking generalities too much here, I think there just aren't very many particularly good ones. As part of the answer for these workloads, I think that you can get a certain distance by taking something off-the-shelf, but you hit a certain point where you can realize radical improvements either in query serving speed, or ingestion efficiency, or in cost by writing your own thing. And we felt like we had the experience from having done this before to kind of jump right to that. And I think for the most part, that has worn itself out. We're pretty happy with what we've got.

And I can be a little bit specific. The kinds of the kinds of benefits that you get versus something off the shelf. Time is a very special dimension, right? As soon as you're using any kind of database that doesn't understand time as a special dimension, you're kind of immediately giving something up for this data and these kinds of workloads. You want something that understands how data needs to be down-sampled, or aggregated, or evolved. How it needs to be indexed over time? So that's kind of one example of a special feature of this data, and this product space, and this workload that you really, really want that idea to be built right into the database at the very lowest level. There's a few more sorts of things like that that I won't go into detail. But that's kind of canonical one.

**[00:15:13] BS:** Yeah. Jeff, one thing that that we also realized with Lightstep was that the tracing data, or structured logging data, which is pretty similar at the storage level anyway, and then metrics data on the other side of it, I think, typically have been thought of as being somewhat incompatible from a storage standpoint. Like you don't see a lot of time series databases that are designed for metrics workloads and for structured event or tracing workloads. And I mean, if you want, or you think your listeners would care, I think we could talk about why that's generally been the case. And it is true that if you just take – You probably can't find a database that's purely built for one that will also work well for the other.

I think what Alex figured out, based on, I think, the experience we had, was that you are able to share almost the entire like query evaluation structure, as well as the very lowest level of the storage, if you're able to kind of swap out certain parts of the engine, particularly around ingest for those two different types of data. And that's allowed us to build a unified query model that works for other types of data without any special casing. And a unified actual – Like the lowest level durable storage is also extremely similar. But then there's this this piece in the middle that can be customized. And we don't have to worry about divergence of those two.

And I was saying earlier about having the tracing data inform other analysis. That basically means you have to be able to do joins against tracing data from other sorts of data, which is something I think people have not been able to do historically, and certainly couldn't do it at Google. And the engine that we have in place on top of this data layer is able to do that because of that, in my mind, pretty clever sort of realization. So that piece I think of what we've done is actually pretty interesting from a software standpoint.

**[00:17:03] JM:** Can you go a little bit deeper into the query path for – If I've got a dashboard, and I'm continually loading metrics, or if I have some kind of standing query against tracing collection, what the query path looks like?

**[00:17:24] AK:** Yeah. I'll leave the standard query part of the question aside for just a second. But for both the metrics data and for much of the trace data, as I said, it's sort of a traditional serving tree. So query comes in, and it comes into some sort of root query server, gets fanned down. We find the right pools for whatever customers making that query. We find all the machines that are involved and we send – We take the query. And depending on what the query is, we kind of chop it into the piece that can be evaluated by the leaf and then the remainder of the query that needs to be kind of evaluated at the root. We tell all the all the kind of leaf serving machines, "Hey, do your part." And they each have a subset of shard of the data. They get the query in, they look at their indexes, they fetch the data. Depending how old the data is, either from disk or from memory, perform their part of the query.

And typically, the output of their part of the query execution is some kind of partial result partially aggregated. Rate computation that's been partially evaluated across the history of a time series. They do as much as they can. And then they bubble up those partial results up to the root that had queried them. And it does kind of whatever reassembly is necessary across those partial results sets. Computes the final answer and bubbles it back up to the client.

So I think none of that is particularly novel at all. There are a bunch of interesting bits in there about, "Okay. Do you have to do some query analysis? What portion of the query can you safely ask each one of these leaves to execute?" And you want that portion of the query to be as large as possible, or you want the leaf machines doing all of the work that they can, because the root ends up being sort of a bottleneck. So there's a lot of interesting technical problems there. But that's kind of the broad shape of things.

**[00:19:00] BS:** I agree that, from an academic standpoint, it's not novel at all. But in practice, there's a huge problem right now, I think, for – I mean, a lot of the open source monitoring things that are out there were designed with like – I think they were designed originally with idea that you have a single very large machine doing a lot of pre-evaluation and storage. And then you'd have like maybe a backing store for some of the super historical stuff. And then you realize that that's not going to work. And so you add other machines and you kind of like manually shard the data out. And that ends up being a very difficult thing operationally to maintain.

So you the solution of doing these partial aggregations, 20th century, think about it in advance. It adds some complexity, but it's sort of doable. But it's a very difficult thing to shim-in after the fact. And this is actually really what led to the creation of Monarch in the first place, is that Borgmon, which it replaced, didn't think about this in advance. And as a result, we had every team at Google running their own Borgmon instances, which was just very painful. Because once it had to be sharded, you had to do that rebalancing manually. And the query evaluation was either incorrect, or inefficient, or both, depending on how it's done.

So this ended up being like a – It's a pretty important and often lacking, I think, aspect of like the query valuation path for time series data. Because what Alex is saying about pushing the computation down, if you don't do that, you have to take a ton of data and stream it up to a single node for evaluation.

And I remember what with Monarch when he made the change to push that stuff down, and it's not like a small improvement. It's like a 20x improvement in the query latency when you make that change, not to mention cost. So it's an important and sort of subtle point, the idea of doing these partial aggregations, because it frees you up to like ignore the difficult sharding problem and make it into an easy sharding problem, and then pre evaluation gets a lot faster. I don't know if that makes sense. Jeff. It's easier with the diagrams.

**[00:20:59] JM:** No. I think I get it. I think I get a sense for it. Can you talk more about maybe the optimizations you've made over time to improve the query latency work or cost optimizations? I just like to hear about optimizations more generally.

**[00:21:16] AK:** Yeah. So it's a lot of – I mean, Lightstep was acquired by a much larger company, by ServiceNow, last summer. And prior to that, we were small and moving as fast as we couldn't and necessarily have a lot of time to do all the optimizations that we wanted to. So there was a bit of whack-a-mole. The customer complains about the problem, we chase it down as quickly as we can and hit it with a hammer. It was a little sort of scattershot.

But in broad strokes, the query workload that you tend to see for this stuff is very bimodal, in the sense that by volume, most of your queries are related to automated, automated colors, whether that's your own alerting infrastructure, right? Your customer comes in and says, “Hey, please alert me when such and such as true. And please evaluate this expression every minute for me." So we're running those continuously all the time every minute.

And so by volume, almost every query is one of those things. And those queries tend to be over very short time slices, right? The most recent five minutes of data maybe, depending on the alert expression. Another class of queries that's completely different, which is a human coming in sitting down at a dashboard or making a manual API call, or whatever, where typically they're looking at much, much longer histories a day, a week, a month, whatever it is, and also typically doing much more complicated operations inside their query.

So there's a sort of balancing act. We want the first class of query to be kind of cheap to execute, because there's so many of them. But we don't necessarily need it to be fast. We don't need the latency to be particularly small, accepting as much as it helps the cost, right? So if your alerting query, if it takes 200 milliseconds to run, or it takes 150 milliseconds to run, you don't see it. You're not sitting there waiting for the results to come back. So we can play some games of sort of trading off, delaying those things if we need to, as long as they get done in sort of timely fashion relative to that one minute cadence.

On the other hand, when you get one of these big dashboard queries that you would kind of know is coming from a human, you want to get that thing done as quickly as possible. And so there's a bunch of games to play there with – You asked us about standing queries. You can pre-compute some of that data to some extent. And there's a game to play about which data to pre-compute, on which data to pre-aggregate, which data to down-sample. And you try to play that guessing game in an uninformed fashion so that most of the queries that a customer wants to render on a dashboard get served out of this pre-computed, or pre-aggregated, or otherwise down-sampled data so that it's faster. And you prioritize those queries ahead, and so forth, as I said. So those are some of the sort of optimization games you end up playing. But I don't want to say it, it's very much a balancing act. It's one of the reasons I like this workload and like the space is that there isn't a one size fits all solution by any means. And the things you want to do for those different classes of queries often tend to kind of get in the way of each other, too.

**[00:23:59] BS:** Yeah. I think, also, the optimizations, one of the advantages of – I don't know, Alex. How many time series storage systems have you built in your life do you think?

**[00:24:10] AK:** It's like four. Five, four, five? I don't know.

**[00:24:14] BS:** I mean, I think a lot of the optimizations are sort of things that are in the past to a certain extent. Like, I know that for – To take the blame or something. With monarch, I had some ideas about like the way we should do certain things early on. And I was probably right about query latency, but very, very wrong about cost. I shudder to think how much Monarch costs to run in this initial incarnation. But at the time, it was kind of like in-memory or a physical spinning disk. Not like SSD disk in the very early days. And so we were really trying to avoid disk for the query path, and managed to fit everything into RAM. But it turns out like it's pretty damn expensive, right? And doing this nowadays, you can rely on SSD if you think about formats and things like that. And I think, Alex, you cleaned up a lot of my mess, some of which after I left Google actually. But that's –

**[00:25:08] AK:** Nope. I came to Lightstep instead **[inaudible 00:25:10]**. Sorry, Google.

**[00:25:12] BS:** It's a very difficult thing to build one of these systems where it's both reasonably fast and reasonably efficient from just a dollar standpoint. And I think some of the stuff we did at Google was replacing a very slow system with a very expensive system. I think what's appealing about the approach that we've taken is that it does split the difference on that, to a certain extent, to improvements networking and storage. It's not like those were available when we did the work at Google. But there are a bunch of important optimizations that I think we could talk about, if you want Jeff, that we did at Google that are relevant to the design we chose with Lightstep and has resulted in like fewer, massive optimization needs, just because the fundamental design, I think, is more appropriate for the workload.

**[00:25:54] AK:** I might add, also, the cost structure of your hardware or your cloud provider I think affects a lot of your decisions. And you just hope to goodness that they don't change the relative costs of things underneath you significantly, because it changes your design decisions. Inside of Google, we were getting sort of – Our compute resources were coming kind of at cost, right? Or even sometimes below cost, because we could kind of slurp up extra unused stuff that other teams weren't using in a particular cluster. That informed some of our decisions about the architecture. When we're out on the outside and you're paying kind of less price for those same resources, you make different decisions.

For an example of that, if you just kind of go poke around the Internet and look for blogs **[inaudible 00:26:32]** time series database, there's a lot of focus in those sorts of things on how should you encode the time series in order to make them as small as possible? And that's all fine. But I think sort of misses the point that if you use a reasonably good encoding and you have some reasonable compression, your disk costs, most of the cloud providers, end up being a minority of your overall costs. It's really the CPU, and to a lesser extent, the RAM, that ends up getting you. And so we spent a lot of time, more time relatively speaking, kind of focusing on minimizing our CPU footprint, rather than trying to minimize our disk footprint. And you can imagine the kinds of the different focus that you would have given that one resources is much more expensive, relatively linear.

**[00:27:14] JM:** When you look at the usage of Lightstep, have you seen any evolution in terms of how customers are monitoring their products and monitoring their infrastructure? Any new sort of observability design patterns over-time?

**[00:27:37] BS:** Yeah, for sure. I mean, definitely, for sure. I mean, there's some things that are in the buzzword category, but they're worth mentioning. I think there's certainly been more of a focus on SLOs in the last couple of years than they were in the first couple of years at Lightstep. And I think it’s okay. The people are jumping into SLOs headfirst without really thinking through some of the issues with the kind of maintenance plan for SLOs, I think. And a lot of times they're being set without much thought to what's above or below someone in the stack. So I think there have been some growing pains in SLO front. But that does affect the design a bit if you're going to make SLOs the starting point for understanding, well, reliability, I guess, at the level of an individual service.

Another thing that I've noticed, and this is something that I'm very pleased about. But in the earlier days of Lightstep, much of the vendor pitch around what would now be called observability was really about how much coverage do you get from an agent? And that was a lot of the differentiation was actually like how much coverage do you get from your agent. And then the analytical piece was kind of trivial and also not that powerful.

And I think there's been a greater need for – Like need-need for analytical power, I think, just because the systems are so complicated, and you have to debug in production and that sort of thing. And so there's been a more of a focus, I think, to deliver actual value on that front, which I think is wonderful for customers. But also, the agent piece is just being totally commoditized by open source and open telemetry in particular. And that's a great thing, actually, really for everybody. I mean, for customers, most importantly, I think they don't take a vendor lock in the way that they decide to get data out of their system, as it used to be that you are very tightly coupled to your vendor. Because if you switch, you're going to lose instrumentation coverage. And it's a great thing that that's not the case with open telemetry. And so we see open telemetry is sort of strategic priority. I think for a lot of the customers we talked with. That's, I think, great for them. It's also good for vendors, actually, because Lightstep didn't kind of fall into this trap. But I think for some of the more like legacy APM providers, they were spending 80+ percent of their R&D resources are going to agent maintenance and agent implementation. And it's a really, really expensive thing to be dumping like smart people into. Especially given that sort of cloud native and stuff like that, you end up with this unbelievably broad surface area of technologies and languages. So the agent thing was just getting kind of unfindable, almost.

So the push towards open telemetry and open source – Yeah, open source instrumentation in general, is something that's changed a lot in the last couple of years. I think is moved the playing field for observability much more towards the analytical side and much less towards the data acquisition side, which, as I said, is being commoditized. I think it's difficult to overstate how big of a difference that's making in the industry. It also makes it much easier for smaller observability vendors to kind of spin up and get started, because they don't want to tackle that problem head on. And that's part of the reason why that space is so crowded right now. But I think that's actually a really good thing. It means there's a lot more innovation and experimentation happening. And people are finding new ways to apply this data. So that's another thing I'd highlight that's changed pretty radically.

I would say, in the early days of Lightstep, before we kind of created the open tracing stuff, people thought it was a bad idea. People are telling us like, “Oh, no one's going to adopt this. It's not going to work.” And I think it totally did work, which is great. But it's had a significant effect on the way that the whole ecosystem has evolved, I think.

**[00:31:12] JM:** So open telemetry, I guess, as you're describing, essentially democratizes the agent and data collection process so that there's less vendor lock-in?

**[00:31:26] BS:** Yeah. I think the mission of the project is to make high quality-telemetry a ubiquitous thing, especially for cloud native software. I say the cloud native not because open telemetry has any particular dependency on cloud native, but more because there is – I think there's a limit to what you can expect to like retrofit for open telemetry. A lot of enterprises are doing some totally ultra-modern Go Lang kub thing. But then it calls down, down, down, down, down. And eventually it's a mainframe, right? So open telemetry support in the mainframe is zero right now, right? So it does have its limits.

The project is maybe bigger than it needs to be at some level. But it has an incredibly broad surface area. So you have SDKs that can be dropped into the actual process itself, as well as an agent like things that can be deployed on the host, as well as the collector. And then a whole suite of protocols, mainly OTLP, the kind of open tracing wire protocol, it's a protobuf thing, that can be used for interrupts between various pieces of infrastructure and open tracing compatible kind of, I hate this term, but middleware for lack of a better word.

I also think a lot of the cloud providers at this point, I mean, Amazon in particular, has devoted a lot of resources to OTEL, open telemetry, but the cloud providers are now emitting open telemetry protocols from their managed services that are closed source. And that's a really important development as well in terms of not just democratizing the data coming out of application, but also coming out of its dependencies. Even like hardware network stuff, like F5 and so on, also like emit a OTLP. So you can get this lingua franca for visibility into a variety of software and hardware that's both managed and not. So it does reduce vendor lock. But it also, I think, provides some compatibility and consistency in the way the data is described.

The thing I think we still need do a lot of work on – Well, logging is just getting started really. Metrics and tracing is in a better place. But the logging stuff is so brownfield. It's going to take time. But then the semantic conventions are like the next piece of this. Like there's a level of actual wire protocols, which I think is going well. But then there's a question like, “Well, what does the actual string attribute you use to describe the name of the process, or the host name?” and things like that, which are fairly simple. But then the diaspora of things you might want to have consistency around is just unbelievably broad.

And for better, or worse, a lot of large companies have already invested a lot of time in their own set of semantic conventions. So there's this long process of just kind of coming up with standard definitions for how you describe the different pieces of the data. But once you get to that point, it allows you to build the next level of standardization, I think, around monitoring itself, where you can have a more declarative approach to monitoring, just like open telemetry gives you a more declarative, democratize as you put it, approach to telemetry. And I think that'll be a really powerful thing. But frankly, we can't do it until the semantic conventions have been stabilized, and they just haven't in their entirety.

**[00:34:27] JM:** I want to talk about a little bit of something that we discussed over email, which is a unified data layer. Can you explain what you mean by a unified data layer and what engineering problems are associated with that?

**[00:34:43] BS:** Yeah. I mean, this is something we kind of touched on briefly a minute ago. I'll hand off to Alex in a second. But when I was talking about how time series data and event-oriented data have historically been pretty segregated actually at the data layer. And I think we have found a way to have them stored in a way that's both efficient and expressive and in a unified way. Alex, do you want to touch on that at all or sort of what – Maybe what the type of approach we had at Google versus what we're doing at Lightstep, just to kind of compare and contrast?

**[00:35:14] AK:** Yeah. So maybe there's three different layers or heights that we can look at this. So just sort of the bottom one is what we were talking about earlier, which is just sort of the storage that the most basic storage and serving layer. The thing that's responsible for keeping the bytes on disk and keeping them intact and giving you back the right bits when you ask for them.

So we talked about before, we have different instances, pools, of the sharded database for different customers, for different data types. But I would say, something like 90% or 95% of the code is the same across these things. And it's this thing we're talking about that's optimized for high-ingestion rates. And indexing is similar across all of these types of data, and so forth. So that's kind of one kind of purely engineering-driven goal, right? You just want a single codebase, a single database set of functionality across all of these things if you can make it work. And I think we've been pretty successful at that.

I can give one really quick example. We used to run Kafka for a variety of parts of our ingestion pipeline. And we're not Kafka experts. We're not Java experts. We run everything else in Go. And we said, we're not good at this operationally. Can we get off of this thing? And we managed to whip together in just a couple of weeks a Kafka replacement using this database that we have built for time series and for traces. We built one that looked like a Pub/Sub. What's the word when you make your Winamp look different? We skinned it. Is that the word for it? I'm dating myself? And it worked great out of the box. So we don't use Kafka anymore. It was really delightful. Because that workload had the same sort of – You got a bunch of pieces of data coming in. They're timestamped. You want them in order. You want to get back the ones you asked. It worked out really great. So we're pretty happy with that just kind of a technology perspective.

One level up from that as the query layer, right? So you want to be able to – When you kind of query these databases, you want to be able to join an aggregate and mix and match the data, right? I want some time series data. I want to join that against some information that might exist only in the traces. And I kind of want that experience to be unified. When I say the caller here typically could be an end customer calling in through an API, or it could be my coworker who builds something in the GUI part of the product calling in, they want that experience to be kind of uniform across the different types of data. So that's kind of the second layer of unification, that uniformity that you might want. And again, I think we have more work left to do there. But it's getting there. And then I think the hardest one, and then I'm going to make you answer this part again at the end. The hardest one is on top of that of the product layer. How do you make these different kinds of data feel uniform inside the experience of the product? And I think that's a super challenging one.

**[00:37:45] BS:** Yeah, I agree that it's challenging. And there's two types of unification that can happen. One is just literally having the same UI components in the same query model from a – Whether it's a simple programming language to do queries, like SQL promt QL type of thing, or if it's a UI, having the same semantics for the two types of data. And I think that's something that we have actually solved, just actually very recently. And, you know, it reduces the cognitive load to learn how to use the system, because you don't have to learn a different way of querying and understanding these two types of data that were typically quite siloed. So that's the first piece.

The second piece, which I think is actually ultimately going to be more profound, but I think is harder to explain, unfortunately, is that – I'll try to explain it by example, because I think if it's done too theoretically, it's just going to be like a bunch of words. If you're dealing with an everyday kind of systems problem, like your CPU just spiked, or your memory usage just spiked, or something like that, that's a problem with the resources in your system, CPU, memory, whatever. And you need to understand why that happened.

I would argue that – I mean, it's difficult to put numbers on this. But let's just say the lion's share of the reasons why your resources spike is that your workload changed, and the workload is tracing data, the resources are not. And being able to pivot from everyday infrastructure spikes to like a principled analysis of how the workload change, and particularly why the workload change, requires a pretty deep join between these two types of data. And that's almost impossible unless you can have some kind of unified data layer like this, at least to do it efficiently.

And yet, it's quite powerful. Like if someone above you in the stack did a software deploy and it turns out that that version explains perfectly the issue that you saw from a CPU spike, what you should be told us like, "Hey, this software just got deployed above you. Here's the before and after. You look at the data. And by the way, here's the username of the person who made that change. You want to get in touch them or file an issue or something like that," right? So that sort of thing.

The hard part is about joining from workload to like infrastructure and back. And I don't know how to do that without unifying the data layer. And I think the reason why a lot of people struggle with that type of thing today is that, frankly, that unification doesn't exist. So you see the spike, and then you go off into another tab and start issuing queries to see what's going on that host or something like that. And at that point, you're stuck in like human mode and not in computer mode. And that's like a very slow and tedious process.

Does that make sense, Jeff? I mean, I think Alex did a good job with the technical piece. I don't know if my piece made sense.

**[00:40:24] JM:** Yeah. No, no, no. I understand.

**[00:40:27] AK:** I would just tack on at the very end there. So at Google, at least at the time. I left and wasn't involved. So we had done a good job at unifying that bottom layer, right? So we had a similar kind of thing where 90% of the of the storage code was kind of similar across these different kinds of data. We had made no progress whatsoever on either of the top two layers, the kind of query integration, query unification or on the product, obviously, above it. And I think that's coming back to something Ben said, this is a place where Google scale kind of gotten the way. We needed to do things that were efficient at such huge scale that it kind of got in the way of doing some of these kinds of unifications. And I think it's a place, happily, where the rest of the industry, Lightstep and others, right? We're not the only people with these ideas. Are kind of actually ahead of where the very largest players are precisely because we can operate at smaller scales.

**[00:41:17] JM:** As we begin to wind down, I'd like to get a sense for what else is on the roadmap? And more broadly speaking, what's beyond the metrics, logs and traces? Is it just improving the UI, and the usage, and the infrastructure around metrics, logs and traces? Or is there some kind of higher-level observability stack to build?

**[00:41:44] BS:** I'll take a shot at that. So first of all, I mean, I think of the logging, the tracing data is converging in of themselves. I mean, almost any log is either about a transaction or about some kind of resource, like a VM starting or stopping or something like that. If they're about a transaction, that should be just part of the trace. I mean, you should stick the context on it, and then it is part of the trace. So there's a convergence happening there. And there's a lot of benefits to that convergence for customers in terms of both features and cost actually, because tracing, the way that you can do sampling and data size reductions and traces, it's a lot more sophisticated than logs, because you have more context.

So I see there's a convergence happening there. It's not going to completely replace logs, but I think it will replace a lot of the logging volume, will be moving towards the tracing workload where you can be smarter about ROI.

Speaking of ROI, I think for metrics, especially in cloud native where you have a lot of micro services, if you look at the usage data – This varies. But it's not uncommon to see the data that's collected is often more than 10 times larger than the data that's created for any purpose. I mean, literally any purpose, and not even getting into whether those queries are valuable.

So you have this massive asymmetry between the amount that's collected and the amount that's used. And to me, this is grievous, and, frankly, customer hostile. I don't think there's been a lot of vendor interest in addressing this issue, because that first number, the data collected happens to be the revenue stream. So it's like solving that problem is pretty dangerous from a business standpoint. But I will say that something that Lightstep I think feels is unsustainable. And I think we're just leaning into addressing it directly. So maybe that sort of is a little bit of a hint about roadmap for metrics. But that's just the data.

When you talk about the application of it, certainly, there's a lot of marketing message about unifying these things. But when you actually get down to it, often, the unification is just an HTML, where you'll take charts that might come from different pieces and put them on the same dashboard. But the unification of the data layer is actually very, very, very early for most players in the space, whether that's open source, or closed source, or whatever. And I think that's an area that we see enormous potential, enormous potential.

Again, I was giving some quick examples. But when that's actually done, I think people are going to spend – You won't have to be an expert in observability to take advantage of the more sophisticated techniques. It'll be much more accessible. And then I also think observability today is something that's only discussed in the context of engineers. The applications we're observing are actually quite diverse, but usually are customer-facing. And in that sense, they have a lot of value to the business in general, and not just for engineers. So I think observability that's able to see inside these applications can also be taken advantage of from a customer success standpoint, a security standpoint, which I think people are starting to move in that direction already. But also, things like finance, and operations, and planning and so on, provisioning, all these things should be taken advantage of observability at the level of individual transactions. And today, they generally don't. So I think observability use cases really ought to broaden beyond the engineers writing the code and maintaining the software.

For what it's worth, the ServiceNow piece that Alex mentioned earlier has a lot to do with that. That's always been our vision, was to make observability useful to as many teams as possible. And I think ServiceNow has built a pretty impressive platform for like a variety of like employee-facing use cases that often are outside of engineering. And I think we see an opportunity to take advantage of that and make observability much more applicable than it is today. But that's more of an outyear thing. And that's not happening all at once. But I'd say in the next several years, I'd expect that to take place and hopefully we can be leading the way on that.

**[00:45:29] JM:** Cool. Well, guys, thank you so much for such a wide-ranging conversation on observability and what you've been up to Lightstep.

**[00:45:36] BS:** It's a pleasure, Jeff.

[END]