**EPISODE 980**

[INTRODUCTION]

**[00:00:00] JM**: Slack is a messaging platform for organizations. Since its creation in 2013, Slack has quickly become a core piece of technology used by a wide variety of technology companies, groups and small teams. The messages that are sent on Slack are generated at a very high-volume and they're extremely sensitive. These messages must be stored on Slack's servers in a way that does not risk a message from one company accidentally being made accessible to another company. The messages must be highly available and they also must be indexed for search.

When Slack was scaling, the company started to encounter limitations in its data infrastructure that the company was unsure how to solve. During this time, Josh Wills was the director of data engineering at Slack and he joins the show to retell the history of his time at Slack and why the problem of searching messages was so hard.

Josh also provides a great deal of industry context around how engineers from Facebook and Google differ from one another. When Slack was starting to become popular, the company quickly began to attract engineers from both of these gigantic companies, both Facebook and Google, Facebook and Google have distinct solutions and distinct perspectives for how they have tackled the problems of data engineering.

We are hiring a software engineer who can work across both mobile and web applications. This role will include work on softwaredaily.com, our iOS app, and our Android application. We're looking for someone who learns very quickly and can produce high-quality code at a fast pace. We're looking to move beyond the world of just being a software podcast into more of a platform of information about software.

If you're interested in working with us, send an email to jeff@softwareengineeringdaily.com. We're looking for somebody who is hungry and wants to learn quickly and wants to build lots of software. If you are that person and you're hungry, it doesn't matter what your experience level

is as long as you have built and shipped meaningful applications. Send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

**[00:02:21] JM**: This episode of software engineering is brought to you by Datadog, a full stack monitoring platform that integrates with over 350 technologies like Gremlin, PagerDuty, AWS Lambda, Spinnaker and more. With rich visualizations and algorithmic alerts, Datadog can help you monitor the effects of chaos experiments. It can also identify weaknesses and improve the reliability of your systems.

Visit softwareengineeringdaily.com/datadog to start a free 14-day trial and receive one of Datadog's famously cozy t-shirts. That's softwareengineeringdaily.com/Datadog. Thank you to Datadog for being a long-running sponsor of Software Engineering Daily.

[INTERVIEW]

**[00:03:15] JM**: Josh Wills, welcome to Software Engineering Daily.

**[00:03:17] JW**: Jeff, thank you so much for having me.

**[00:03:20] JM**: The data infrastructure at a given company, it usually starts with a transactional database. You have a Mongo database. You have a MySQL database.

**[00:03:28] JW**: I hope you don't have a Mongo database. But otherwise, yup. Got you. I'm unemployed. I'm going to throw all kinds of shade during this podcast. Yeah.

**[00:03:35] JM**: All right. We're getting people normalized, unlike a Mongo database.

**[00:03:39] JW**: Oh!

**[00:03:40] JM**: So the transactional database at the startup, you got a couple of people. You only have one database and it's just storing all the messages or storing all the comments or storing all that ridesharing activity, all these different things.

**[00:03:54] JW**: Orders. Sure. Absolutely.

**[00:03:55] JM**: Yeah, the orders. Company starts to take off. You start have tons and tons of these transactional data transactions happen and the database gets really big. That's not a problem. We know how to scale those things. Eventually you get to a point where you want analytics as well. You want perhaps a data warehousing system. You want perhaps some sort of data Lake. You want these ETL or ELT systems.

Could you contrast the type of operations that take place in that company that is the purely transactional company versus a company that has advanced enough to need these kinds of analytics or OLAP systems?

**[00:04:37] JW**: Yeah. Okay. I think from my point of view, it fundamentally comes down to a hiring thing. Let's take for example Slack where I used to work. When I got to Slack in 2015, it's like obviously growing like a weed for a couple of years at that point. Not all of the analytics, but like the vast majority of the analytics were still run against like a replica copy of the transactional data system. We had like a job queue infrastructure that we had built. By we, I mean, the people who were there before me, since obviously I wasn't there at that point, that Slack had built for doing asynchronous operations, and obviously ETL can be thought of as an asynchronous operation.

Yeah. When the first kind of set of analysts got there and they were dedicated to answering business questions, essentially, full-time is their job, their only real option to do that was to like spin up their own little database that had a cached copy of the results of ETL calculations that were run against like replicas of the transactional database, and that was the system.

I think for companies that are in this situation that are like thinking about making this transition, for sort of this chicken and the egg problem, right? Essentially, should you hire an analyst first or should you hire like a data engineer first? Slack, I think like a lot of companies sort of default

towards hiring the analyst first and the analyst gets there and kind of has to work with what exists.

Then eventually you manage to hire data engineers and the data engineers start doing the work to spin up obviously like the data warehousing systems and the ETL or sort of the – Whatever, the extraction process and replication process, whatever, whatever, for moving data out of transactional systems, restructuring it in a way that's optimized for analysts and basically doing the things that makes the analyst's life much better, and fundamentally allows you to scale out your analytics team, where it's not the case that every analyst must have an in-depth knowledge of the transactional system. It has to be really, really good and really, really careful not to say accidentally dropped a production table, which happens.

Yeah, really for me it's really this kind of hiring thing. If you've reached a point in your growth in the success of your company, whatever, whatever, where it makes sense to have like a full-time analyst, that's kind of how you get things going. If you grow further and are more successful and need to scale it up, building out that analytics infrastructure, that analytics – I cringe a little bit as I say it, value chain requires these additional people or, honestly, like these days, more and more just like between Segments and Fivetran, and I don't want to leave anyone off the list, but there's like a ton of these different companies that are really providing a lot of the very like basic common data engineering functionality as a service. You can really actually get a long way without even having to hire anyone. You can hire these companies to do it for you.

**[00:07:29] JM**: Wait. The company does it? Don't you need some person at the company who's going to stitch – Like you need a person at Slack who's going to stitch together all these different solutions?

**[00:07:37] JW**: Definitely. What I mean is you don't need a specialist per se. You can have a reasonably qualified generalist software engineer who knows how to like stand up APIs and stuff like that and have them provision and run these systems to do most of the work. You don't need necessarily like dedicated people. In particular, you don't necessarily need like a specialists per se. Someone who specializes, or a team of people who specialize in this sort of data infrastructure, data analytics pipeline.

It's all those tricky things. There's a great – Joel Spolsky has this great essay, and he has a lot of great essays, but the one I always go back to is the law of leaky abstractions. It's this essay about how we've created these layers and layers of abstractions on top of all the different things we built. Even as the attractions like save you time, they don't necessarily like save you the cost of understanding the kind of intricacies and the aspects that like the abstractions or abstracting over, if that makes sense. I find that eventually – I'm trying to think of a good example. Let's say garbage collection is an abstraction that everyone is familiar with, right? We use garbage collection so we don't have to manually manage memory, because that's obviously stupid and a waste of time.

You can use garbage collection without having to understand garbage collection. But if you run a system for long enough and it gets kind of interesting, you will inevitably run into a situation where say your garbage collector is like pausing the world and your whole system just seizes up for a couple of seconds. At that point, to figure out what was wrong and diagnose it and remedy it, you are kind of forced to understand like the intricacies of the garbage collector and like what exactly it is doing at some level.

The garbage collection saved you all this time until – But like the check comes due in like a later date in a lot of ways. I think what's hard for folks getting up, like spinning up data infrastructure to support analytics data science, machine learning, whatever, whatever, is you're sort of immediately confronted with like the fact that you don't really know what's what. You don't like know all the things. You don't know all the pieces you need to know. So you're kind of immediately, even as you like can use these tools to save yourself a ton of time, it doesn't save you the cost of like knowing what to do, if that makes sense.

**[00:09:50] JM**: Yeah.

**[00:09:50] JW**: Yeah.

**[00:09:51] JM**: Slack specifically, when you joined, was there analytics? Was there OLAP? Was there an offline nightly Hadoop job running? Give me the state – Because we all know how Slack transactional processing works. We all know like – I've done some shows on that, but the root of it is like you have a message that gets sent to people and then the message gets shared

with the people who are on the channel, and data gets written into some kind of scalable database and you got all kinds transactional issues there that's really interesting.

Then as far as the OLAP infrastructure, the things that I can immediately recognize are that building a search index overall this text is nontrivial, and that almost sounds like an online analytics processing kind of job. Not really an offline, because if you send a message, you want it to be indexed very quickly.

Yeah, I guess just give me your overview of the analytic processing, the perhaps offline or non-transactional processing that existed when you joined Slack.

**[00:10:57] JW**: I joined Slack in October of 2015. Slack hired their very first data engineer in July of 2015, like a little bit before me. The person they hired, and I'm going to protect his privacy and not mention his name, is honestly a phenomenal engineer, like one of the best engineers I've ever worked with. I changed my mind. I'm going to mention his name. It's San Babourine. He's absolutely an incredible engineer.

He had set up almost everything kind of by the time I got there to like nominally manage the team. I had to make a few – I had made a few different tweaks. We learned things as we went along. But broadly speaking, we ran a Netflix-style OLAP data architecture in the sense that everything, all of our data, all of our logs and our sort of like replicas of production database tables and stuff like that were all stored in S3. We used Parquet as our file format of choice. We ran EMR clusters almost exclusively, and this was like really back in the dark ages of EMR. This was like EMR 3, EMR 4, which was just an absolute nightmare for reasons I can get into later. It's much better now. But it was Hive primarily for ETL, and Presto primarily for interactive queries on top of it.

Then we ran ERPAL, which was a sort of short-lived version of this – It's this thing Airbnb built. It was a clone of a systematic at Facebook call HiPal for doing kind of query dashboarding visualizations, like very lightweight, very kind of primitive sort of stuff, but it was fine. It's been obviously, again, replaced by much more sophisticated things at this point. That was what we ran, and we initially started querying over all of our replication logs, Apache access logs, all that kind of like the great stuff that is fairly standard. Then eventually built out systems for doing

richer application logging using Thrift as kind of our schema definition format, transferring Thrift records into Parquet, and then eventually built an entire system stand.

I should say, really built an entire system for spinning up basically replicas or backups effectively of our production MySQL systems and then just dumping them all into S3 every day. It's a system we called Scooper. It's essentially a very clever orchestration layer around Apache Scoop, although at this point I think we actually ripped out Scoop. Scoop ended up being the slowest part of it at some point. Anyway. Yeah, that was kind of the foundation of what we did. I guess that it was very Netflix style. We treated our Hadoop clusters as these ephemeral things. We didn't really care if they died. We would just spin up a new one, like annoying, but like not the end of the world.

We had a homegrown ETL engine that Stan wrote that after, honestly, one of a really very bloody battle, we eventually replaced with Airflow. Airflow like 173, which I think that's a whole other – We can do another hour on like the intricacies of Airflow. Airflow to do kind of most of our ETL stuff. That was the basis, that was the foundation. We built all these stuff, and I think I've talked about a little before that we sort of ended up building kind of like a ghost city for a while. Stan and I were really just like laser focused on building this stuff up for like maybe six months or so and sort of why we're building all these infrastructure. The existing analytics team was using the existing analytics infrastructure, which was the ETL jobs running on our production systems using the production job queue and then kind of piping stuff into a cached kind of analytics database.

It was funny. It really took us a while. It was not until we started like aggressively hiring analysts, data scientists, engineers from the Googles and Facebooks and Twitters of the world that we actually got the people in who knew how to build – Knew how to use the stuff we were building, basically. Anyway, it was – Yeah.

**[00:14:34] JM**: Can you shed more light on that? I've seen plenty of conference talks and KubeCon sessions and podcast and stuff. Isn't everything there? Don't I know how to build data infrastructure by just watching the videos and those kinds of things? What can somebody who's worked at Google and Facebook tell me that I don't already know from looking at software architecture diagrams?

**[00:14:58] JW**: Oh! I'm sorry. I guess I'm not being clear. The infrastructure Stan and I were building was fundamentally like – It was basically like we're using Facebook, fundamentally, Presto, Thrift, Hive. All the things we're talking about here are fundamentally like Facebook-based technologies. I was a Google person. I spent four years at Google and I kind of learned how to do things the Google way.

We ended up building kind of like Google's style data infrastructure within the Facebook technical ecosystem, which caused like a number of comical problems from happening too.

**[00:15:32] JM**: You got to tell me more about that.

**[00:15:33] JW**: Yeah. It was just really – It's absolutely terrifying.

**[00:15:34] JM**: How do these things even differ? I thought they were the same thing.

**[00:15:37] JW**: They're really not. They're really not. In very important and fundamental ways, they're really not. You're right. It's not widely talked about, but for those of us who lived it, which may be me and a few other people, they're quite subtly different in ways that turn out to be incredibly, incredibly important and very painful to remedy.

What I mean is we didn't – No one at Slack knew how to use that stuff. We invested like zero time in training these. What do we have when we got there? We had the existing analytics infrastructure? We had Logstash, right? The ELK stack. So the engineers use ELK and like Grafana and whatnot, because that was what engineers knew how to use. The analysts used kind of like their MySQL system that they built themselves and they kind of new how to use. It's not until you hire like the Google or Facebook people who come in and are like, "Okay. We're here. Where's the Presto thing? Where's the Hive thing? Where's the Dremel thing?" Where are the tools I know how to use? What are they called? Where are they?"

That we actually started getting some traction with this system, because the people who were like there, they had their tools. They knew how other tools works. They had problems to solve using the tools they knew how to solve and they were like – You're basically saying, "Hey! Look

at this entire new, elaborate technical infrastructure that is incredibly powerful, I promise, I swear, but you know absolutely nothing about. You don't know how to solve the problems you know how to solve using these tools. You don't know what problems these things can solve." Blah-blah-blah-blah-blah.

Whereas when you bring in the people who've worked in those worlds and know how those worlds operate, you really just have to tell them, "Yeah, yet it's called this. Here's the URL. Go nuts Knock yourself out." That was sort of the thing.

**[00:17:13] JM**: Clarify to me what is it between the Facebook stack in the Google stack.

**[00:17:17] JW**: Yeah. It's a great question. So where to begin? Slack was my first real experience working with like a ton of Facebook people, and I think one of the problems of Google people and Facebook people, at least people who've only worked at one or the other, is they think that the way that things were done at Google or Facebook is like the way, capital T, capital W. It is the truth. It is the received wisdom. There are elements of that that are correct, because there are things that Facebook does and things the Google does are fundamentally the same or have like a very least the same underlying principles associated with them, but there are lots of other things they do very differently. This is true across their entire stacks.

Generally speaking, there are consequences to a lot of what are effectively like random choices. I will give you an example. This is my favorite example. At Google, for a very long time, there was one – Google uses protocol buffers, Facebook uses Thrift. There are minor differences between them, but they're, again, pretty inconsequential, at least in my experience, except when they're not. Nevermind.

But anyway, they both use – Facebook uses Thrift for RPCs and for logging. Google uses protocol buffers for RPCs and for logging and they have for a long time. At Google there was a protocol buffer called GWS log entry proto, GWS, GWS, stands for Google Web Server. It's Google's homegrown web server. GWS log entry proto, and it is this gigantic protocol buffer that contains like absolutely every single log field for any Google service anywhere ever. It is this gigantic monstrosity of a thing, and it turns out to be like fairly useful in a bunch of different contexts. They have like one giant log record to rule them all, because you can kind of assume

that whatever service you're spinning out is going to have a GWS log entry proto. You can assume that everyone's going to have this globally unique identifier associated with it. You can assume you kind of glue these things together in kind of neat ways. There's a lot of power that comes from this.

When I got to Slack, that was the way you did logging. So that was what I created at Slack. I created a Thrift record, again, doing the Google style thing in the Facebook infrastructure called the slog. A lot of that is just because I am super bad at naming things, and anyone who ever works with me should never allow me to name anything. That is like my one take.

**[00:19:33] JM**: What's the name of your child?

**[00:19:34] JW**: His name is Wesley.

**[00:19:35] JM**: That's not bad.

**[00:19:36] JW**: That's not bad. It's funny. My wife and I, we didn't know what the sex was going to be and we're really hoping to have a girl. This is funny, because you will obviously hear this at some point in the future. You're going to telling this story.

We had girl names picked out. We had like kind of like a few nominal like boy names kind of picked out, but when he was born we realized that like none of the boys names fits, and we're going to call him like Soren, or  Case, but it's just like you met him and he's only one-day-old, but he wasn't a Soren. It's like, "This is not who this – This is not a Soren." It took us four days actually to come up with his name.

**[00:20:09] JM**: Wow!

**[00:20:09] JW**: Yeah.

**[00:20:09] JM**: Okay. Anyway. Sorry. Slog. I think it's a good name.

**[00:20:11] JW**: Really important sidetrack.

**[00:20:13] JM**: But let's move on to slog.

**[00:20:14] JW**: Slog. It just got worse. I started doing every –

**[00:20:17] JM**: Slog. This is the single log message to rule them all for Slack.

**[00:20:20] JW**: Yeah, effectively. Like the Slack log, the single log.

**[00:20:23] JM**: Meaning that whenever the developer says like service.log, it just emits this thing that has everything in it?

**[00:20:30] JW**: Yeah, basically. This was kind of like the container's like Thrift record for every other log demand. The thing is, it's not just for like sort of Slack's core web service, like the web application, like the monolith of the core of Slack. It's like everything is a slog. Every service spits out slogs. This is like the Google way. Every service, including services at Google that are not GWS with a handful of exceptions spit out a GWS log entry proto. Every service at Slack spits out a slog. Again, few exceptions, but generally speaking that's the way it works.

That is not the way they do logging at Facebook. Facebook uses I think a much simpler system, many ways is vastly better to be honest with you, which is kind of like – I eventually created a Facebook style log at Slack. Again, this is sort of more evidence for the I am bad at naming things argument or proposition or whatever, call the clog. Yeah, I know. You're sort of like – You guys can't see it, but Jeff is like literally throwing up into a wastebasket right now. The clog.

The sort of the core of Facebook logging is like every service has its own logging. It's like utter lawlessness, but generally speaking, every service has like – Every log event has an event ID, which tells you some idea of like what is this thing. What is this event? The idea of like why did Google build this sort of Uber crazy not so log it was because they wanted to kick out one log at the end of every single request, if that makes sense. Whereas Facebook was running – That was because Google was like compiling stuff. It was all C++. It wasn't really a big deal to have like state around that you could kind of persist across different requests and so on and so forth. That was how they did things.

When Facebook did logging, they did lots and lots and lots and lots and lots of events, logged events per request. As the request goes along, they would just fire off like an RPC call or an HTTP call to a local scribe server containing whatever information they want blog right now. Instead of having like one gigantic source of truth for the entire request, Facebook have lots and lots of little tiny ones. There are lots of downstream consequences to this decision, and the choice was honestly in my opinion just a function of the fact that Google was born at a time when like Kafka didn't exist.

Google was using like, essentially, like a log rotate-based logging system for their application servers for a long, long time. It was the world's best log rotate system, but it was just log rotate. Whereas PHP, stateless, all these kind of other sort of consequences the way PHP works, every PHP request starts like de novo nothing. There's no state. There's nothing, right? There're actually some really great consequences to this.

Anyway. So they opted for like lots and lots and lots of tiny events over the course of the requests. This fairly like arbitrarily seemingly inconsequential decision has just tremendous downstream impact on like every other thing you do and how you build infrastructure and so on and so forth. There was also a significant element of like the business they were in, sort of driving the way they thought about data modeling.

Google search, and search, there is not like a database generally speaking. There's no relational database. It was just a search index. So at Google, the log is the source of truth, and the logs are almost always highly, highly, highly de-normalized in the MongoDB sense of the term where there's like an entry, like a repeated array of complex records, which have information on every search result and so on and so forth.

Whereas Facebook, again, with these lots and lots of small very simple events that are just basically like simple key value pairs and then their existing relational infrastructure since Facebook is obvious like a very big and elaborate MySQL system, relational tabular thinking and relational tabular data processing was really fundamental to the way they worked.

Google was kind of born in a non-relational world and developed non-relational data infrastructure tools, most famously MapReduce. Facebook came along with like a very relational oriented system and are in fact their original like data warehouse was essentially like Oracle and stuff like that. Hive came along later. So it was kind of an interesting situation where Google is born kind of non-relationally. Invents all of these very clever, brilliant non-relational infrastructure, comes along later and slaps some SQL on top of it via Dremel and Tenzing and other systems they built.

Facebook has this kind of relational system, grows and grows and grows to the point where they realized they need like the Hadoop-ish kind of clone of Google stuff and then creates like a SQL layer on top of that, which ends up becoming Hive to kind of keep their data model on their way of working consistent across their entire environment.

I think the decision at Google to create like GWS log entry proto and do it the way they did was probably something that like an intern named like Kevin, or Susan, or whatever created like one day in 2000 thinking nothing of it that had this just an enormous downstream ripple of consequences. The thing to remember is from my perspective, showing up at Google in 2007, 2008, I can't tell because all of these stuff has grown up around us, right? There are all of these sort of subsequent decisions that are coming, and I didn't see it happen. I don't know the history. I don't know this stuff. To me, it's all just truth. It's all just received wisdom. This is how we –

**[00:25:44] JM**: We need an episode, the history of GWS log entry.

**[00:25:48] JW**: I would be happy to. Again, I am unemployed. I have nothing else to do, and I love clearly talking about this stuff in a way that I think I maybe had forgotten over the last couple of months.

**[00:25:57] JM**: Well, we need to find Kevin or Susan or whoever it was.

**[00:26:00] JW**: I would love to.

**[00:26:00] JM**: Who did that.

**[00:26:01] JW**: I'm sure they're in witness protection at this point. Eventually, [inaudible 00:26:04] a couple of years ago, Google actually like finally hit the scaling limit of GWS log entry proto, where there was so much –

**[00:26:11] JM**: There are too many fields.

**[00:26:11] JW**: Yeah, and kind of like in the Android sense, where like Android has this limit where you can only have like 64,000 functions or something like that in a binary. There's some weird limit around this. Same thing happened with the Java compile version of the GWS log entry proto, where it like exceeded the capability of the Java compiler. This kind of stuff happens at Google all the time. They handled it, but it was just kind of like the bill came due. It just came due. After 15 years –

**[00:26:39] JM**: You're sick, man!

**[00:26:41] JW**: I'm sorry. It's just this is – I don't know if this is the nature of the terrible industry that we've chosen to work in. It's just like what we do to ourselves. Human beings, man.

**[00:26:50] JM**: Right.

**[00:26:50] JW**: Damnedest thing.

[SPONSOR MESSAGE]

**[00:26:59] JM**: Redis is a fast in-memory database system. Engineers have been using Redis for more than a decade because of its reliable object caching, but that's not the only use case of Redis. Redis can be used as your operational data store for queuing, streaming and other data applications.

We recently had an episode of Software Engineering Daily with Alvin Richards of Redis Labs in which Alvin described the use cases of Redis, and I enjoyed learning about the flexible architecture and how Redis uses memory and persistence to create an API that solves a variety

of problems. You can listen to that episode or you can go to redislabs.com/sedaily to find out about how Redis can help as a data layer for your microservices.

Redis Labs is the company that makes Redis Enterprise, which offers performance, reliability and professional assistance with your Redis instances. If Redis is on the critical path of your application, go to redislabs.com/sedaily and learn about Redis Labs as well as some of the design patterns for Redis that you might not have seen before. That's redislabs.com/sedaily.

Thank you to Redis Labs for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

**[00:28:22] JM**: We've gone very deep on this.

**[00:28:24] JW**: I know. I don't know why.

**[00:28:25] JM**: Esoteric subject of how Google data infrastructure and Facebook data infrastructure differ from one another through the lens of the differences in how the logging infrastructure developed or the log message infrastructure. I understand there were downstream ramifications, and so you have a divergence in the system design there. Then at Slack, your post-Google, your post-Facebook, timeline-wise, and you've got engineers from both of these places coming in and they're discussing how should things look here at Slack.

I mean, if you've got these two past systems where things were one way in the Google world, things were another way in the Facebook world. How was a resolution reached as to how things should be done at Slack?

**[00:29:18] JW**: Yeah, it's a great question. The honest answer is that when I got there, I made essentially all of the decisions about how things should be done at Slack. I made a number of them, and I was coming like largely from a position of, again, this sort of myopia of not understanding the evolution of these systems and really like not understanding of the decisions I was making, which of them were based on principles and which of them were just sort of like

path dependent consequences of random decisions that were made earlier in time, if that makes sense?

When the Facebook people got there, they were mostly like angry and confused. Why I was doing things or why I decided to do things the way I've done and not being like the world's most mature or enlightened person either today and certainly not four years ago. I would say I was not overly sympathetic to their concerns, probably speaking. I'd say like, "Well, that's Facebook. It's great. Y'all are obviously very successfully. You've done well. But I'm from Google. This is how things are done. Just trust me." It went over about as well as you'd expected to, like all that kind of stuff. My education was hard and difficult, but I would say very well learned at this point.

Anyway, for some things, I would say when a lot of like sort of the choices or the way things were done, we're generally like a function of the first mover. There was too much to do. There was a million things to do. There was no shortage of things to do. Everything needed to be done. We were doing things as fast as we could and we were just trying to keep the lights on the vast majority of the time. Just keep the data flowing. Keep the pipes running, that kind of thing.

Generally speaking, if like a Facebook person came along and said, "Hey, there was this thing that was really useful at Facebook. I think we should build it here." I was like, "Great! Knock yourself out. Just don't talk to me for the rest of the week, and that will be fantastic," which is, again, as the director of data engineering, was not like the right response, but neither here nor there.

A lot of those things were just like phenomenally great and incredibly powerful and taught me a lot, and I think very much taught me a way it's possible that some of these Facebook people kind of like maybe know what they were doing and had like some pretty good ideas about how stuff should work. It's a lot easier. I don't know if it's better, but it's certainly a lot easier. Go without saying that I would do a lot of things differently had I – To do over again, so many things differently.

The irony of course is that in doing these things differently, no one would appreciate what I'd done because they hadn't had the experience of not having it, if that makes sense. The terrible

analogy I like to use is no one appreciates the person who on September 10th, 2001 mandated that every single airplane cockpit door had to have a lock on it, right? We don't appreciate these decisions, right? You can't really appreciate all of the pain that something saves you the vast majority of the time, right? That kind of stuff. You just take these things for granted. This is just like the way it works, and of course it works that way, and that's fine.

I might joke, I think my tweet was something like when you're an early technical person, and this is just vastly more true I think for like CTOs and the very early, early technical employees at Slack and other places, is you make an enormous number of decisions. The vast majority of them were correct and good, and if you hadn't then, guess what, you wouldn't be in business anymore, right? But those good decisions are like invisible. No one can see them. They're taken for granted. Of course, because you're a person and you're imperfect and you can't see the future, you make a small number of like bad decisions. Some of them are like really, really bad. Those bad decisions cause pain and suffering for everyone forever and ever one like kvetches about them and all that kind of stuff. That's your reward. Congratulations. Hopefully you'll get a bunch of money to go with it, I guess, when your company goes public. But that's the deal.

**[00:32:59] JM**: It almost seems like the decisions didn't matter that much.

**[00:33:03] JW**: Some of them do. Some of them don't. Some of them do. Some of them don't. Yeah.

**[00:33:06] JM**: But you can always compensate for them, right? You always just figure out some architectural way to compensate for them and sometimes that architectural way of compensating for them ends up being a strength of the company later on.

**[00:33:18] JW**: You can always compensate for almost any bad decision with sufficient money, like broadly speaking. I mean, I think Airbnb is I think in many ways like a phenomenal case study. I mean, they essentially just like literally airlifted an entire Facebook data team over to their company around 2014, and the team came in and – I mean, you should get like – I actually recommend some Airbnb people to have on the show, because they're fascinating. It's like one of my favorite story.

That was great and that they just like – Because, I mean, they were like – It was a disaster. It was an absolute disaster. It was just – Anyway –

**[00:33:45] JM**: Well. Hey, man. You get a bunch of designers building a company. You get the worst and the best of that.

**[00:33:50] JW**: I mean, I think there's a lot of truth to that, and that's just the way it goes, right? The Facebook people came in and they re-built like the Facebook data style. I like said, that solves a lot of problems. There's none of this nonsense conflict about like should we do like sort of these ridiculous – Do we eat our eggs with the top – Whatever, the Gulliver's Travel thing. Do we eat our eggs with the top or the thin end, or the thick end, whatever? Kind of religious war, right?

**[00:34:11] JM**: I was with your analogies until then.

**[00:34:14] JW**: Until then.

**[00:34:15] JM**: You lost me there.

**[00:34:15] JW**: This one is in Gulliver's Travels, like Jonathan Swift is parodying this idea of like some kind of religious war between two groups of people. One of whom eat their eggs, like crack their eggs on the small end, and some of them do it on the bottom end. I don't know, the top, and this is like a battle. This is a lot of like –

**[00:34:32] JM**: For soft boiled egg.

**[00:34:32] JW**: Yeah, for a soft boiled egg. This is a lot of like the religious wars. We have our own technology choices are fundamentally like this. They're kind of like stupid. Skipping all of that, awesome. Saves you a lot of time. That's great a lot of time. A lot of heart ache.

One of the things they didn't do and cause them an enormous amount of pain was they used JSON logging for years and they did not replace the JSON logging until I think a couple of years ago is my understanding. They finally moved over to Thrift.

**[00:34:59] JM**: And that's problematic, because it's going to be slower. It's going to have more parsing and de-parse, or whatever, transcoders or whatever.

**[00:35:09] JW**: Sadly, this is like a whole other hour-long topic on the problem.

**[00:35:11] JM**: Okay. All right. Let's not get there.

**[00:35:13] JW**: I'll do it briefly. The human cost is the problem. The human cost is the problem. JSON on enforces essentially no consistency. You can name things whatever you want. You can use camel case, you can use underscores, you can do typos. You can do all these kinds of things. Your application logs, which are obviously very useful for lots of things have essentially no structure and no reference for like what the fuck is in them.

To compensate for this, you hire – Facebook did this too for a long time, armies of data engineers who effectively become the schema. They become the schema. You didn't have like a Thrift record and impose a small cost on everyone whenever they want to log something. They have to update this stupid Thrift record, right? This stupid Thrift schema to say what they're going to log, or you can utter lawlessness and have like a data engineering team that needs to scale with the rest of your application engineering team to deal with all of the vagaries and subtleties of all the different logging. That's what Airbnb had for a long time, because you have to, because otherwise you go nuts. You go insane. They fixed it. It's really, really hard. It takes a long, long time. It's incredibly expensive to do. Human capital time, all these kind of stuff, but it can be done.

**[00:36:24] JM**: Let's refocus on Slack.

**[00:36:26] JW**: Oh, God! Do we have to? Okay.

**[00:36:27] JM**: From 2015 to – You left in, I guess, this year, right? 2019?

**[00:36:32] JW**: Yeah. I left. I'd say November 1st was my last day.

**[00:36:34] JM**: Over that period of time, was there some particular technical problem that was so hard to solve that it's very memorable or there's a particular lesson you can draw? When you think about that question, is there is some singular event or architectural feature of Slack that you had to fix that comes to mind?

**[00:36:54] JW**: Oh, that I personally had to fix?

**[00:36:54] JM**: Yeah.

**[00:36:56] JW**: I'm sure there were like dozen of them to be honest with you. I think the one I am most familiar with was rebuilding Slack search, like that was really – That was the hardest, and in many ways like the best thing I've ever done, rebuilding Slack search. That was really, really, really hard and very, very rewarding.

**[00:37:15] JM**: I saw your talk on that.

**[00:37:15] JW**: Yeah, you saw the talk at Lucidworks or whatever, whatever their – Activate, whatever it's called. Yeah. That was best working experience of my life. I will never top that, I don't think. If I had to guess my work for another 20 years, I don't think I will ever top that working experience. Yeah.

**[00:37:32] JM**: Why isn't that just like building – I mean, I think we did actually this show about search at Airbnb, but Airbnb has tough search problems. Every company has tough search problems. We can take Elasticsearch off-the-shelf, or take Solar off-the-shelf or whatever, call the Elastic folks.

**[00:37:50] JW**: Call Lucidworks. Yeah, that's right.

**[00:37:52] JM**: Call the AWS Elasticsearch service team if you're on a budget.

**[00:37:55] JW**: That's right. Algolia is good people. I mean –

**[00:37:57] JM**: Algolia is good people.

**[00:37:58] JW**: Lots of good options.

**[00:37:59] JM**: Why can't you just do that? What's so hard about it?

**[00:38:02] JW**: Yeah. It's a great question. Most people can do that. You're absolutely right. The vast majority of people can. I want to give some context for this. I picked on Mongo a little bit at the first part of the talk, and that's unfair and cheap and I'm a bad person and I apologize for that. I don't apologize for being a bad person, but I apologize for the cheap shot at Margo.

But the thing I really liked about working at Slack, technically speaking, it was a very boring by the book company. It was development by a bunch of X-Flickr people. When I first got the Slack, it was kind of cool. It was like my second day there. I had kind of engineering 101. This is how Slack works. By the end of it, I could pretty much just start working. There was not much to learn. It was like not much to it. It was really straightforward.

You could have pick up an engineer from Flickr in 2005, dropped them at Slack in 2015. They pretty much would've been able to work, right? It was like PHP, Apache, MySQL, very, very boring, no frameworks, no magic and I would say just lacks credit, I think in a lot of ways. They were very much focused on like boring – Dan McKinley, McFunly, has written about this, like choose boring technology, and in my opinion, completely right. I think like a lot of my nightmares at Slack were really imposed, like I did them to myself when I would choose like the bleeding edge version of Spark where like, "Oh crap! I'm getting an error message, and there's no stack overflow answer. Oh God! Oh no! I have to go figure this out myself. What have I done to myself?" That kind of thing.

By choosing boring technology, you avoid all that. All the answers are known. You can pretty much call anybody, right? The same choice was made, the same sort of approach was used when Slack was building its search infrastructure. Again, we've talked about this a little bit in the activate talk. But Slack search was built in 2014. It was built entirely on Solar. That was what the engineer who had worked at Flickr, and actually I think amazingly was even in a high school band, was Stuart at one point, and he knew. He knew how to use Solar. It was kind of a Solar 4,

because that was what existed at the time, basically solar cloud was still like not quite really a thing. Solar 4, team-sharded indexing system. That's what it was.

I think I thing in early 2016, when we started building out what was then called SLI, search learning intelligence, primarily in New York, we started hiring – We hired a lot of like Foursquare people, because that was where Noah Weiss, who the head of that group was from. We hired a bunch of Foursquare people, a bunch of etsy people, a lot of search experience both at Elastic and Solar coming from those places, obviously, that kind of thing. We set out to start rebuilding Slack search on top of Solar Cloud instead, instead of Solar.

We made a bunch of choices One of the choices we made was to – First of all, we made the choice of stick with Solar Cloud as supposed to Solar, and that was a very consequential decision in ways we couldn't quite fully understand. But the idea was that if we just migrated from Solar to Solar Cloud, we didn't have to rewrite any of the query construction indexing layers in the application itself, because like you said, that is all somewhat real-time. It's handled by that same job queue infrastructure to do the indexing, and of course the querying constructions real-time.

We thought, "Okay. It will be easy to migrate from like querying the old Solar clustered to querying the new Solar Cloud cluster, because we don't have to rewrite anything. We just have to point in to the new cluster, and that was like largely true, and that was great. We opted to build the entire sort of like historical index basically offline using like a MapReduce pipeline. Again, a lot of that was just like the availability of resources. We had like –

**[00:41:37] JM**: By the way, you have to build an index four every single team, every single Slack team that gets created.

**[00:41:43] JW**: Every single team that gets created. I think as a – I don't know if this is still true, but again, when we were doing it, every message for every team is available in the index with effectively the same like quality of service so that if you decide to pay for Slack and you turned it on, boom! Your search is on. You get everything right there. Just like that, magical. That was the idea. That was the vision.

Slack ingests hundreds of millions of messages per second. The query volume is maybe like 1/100th of that, right? Probably speaking. Unlike say an e-commerce search or like Airbnb or whatever, events index messages are coming in much, much, much, much, much, much, much, much, much faster than queries are, and this has like – This is basically like Elasticsearch style querying and structure and stuff like that, right?

Slack is also – At this point, it's a little fuzzy. The three largest Solar Cloud clusters in the world are Apple, Salesforce, Slack and Redit, are the big four. I'm sure there's like a few others that are like creeping up since then. But at the time, they are basically like in a class by themselves. You do not run Slacks like multibillion document, like message indexing cluster the same way you run like a 50 million document e-commerce indexing system. It's just a completely different sort of – The funny thing, I think –

**[00:43:02] JM**: You're saying because the writes are just so much faster in a system like Slack than in an e-commerce store where, whatever, some sellers send you their new things to put on the side. It's like, whatever. We could figure how to do that. You just have such high-volume with a messaging system.

**[00:43:19] JW**: That's right. With a messaging system, the right volume. It's just so off the charts. The same way like a logging system has effectively off the chart write volume compared to read volume, stuff like that.

**[00:43:28] JM**: Just for a little bit of context for people who haven't thought about search, every message that comes in, so let's say it's a message that says, "Hello. I am Josh." Anybody who searches for that message needs to – The lookup needs to hit all of those different strings that are in that sentence, and in order to have that kind of lookup system, all of those different tokens or words or whatever need to be basically a look up-able. They need to be key eyes and so that the value of that actual message can be looked up on the backend infrastructure, and the process of doing that breaking up of the string and turning it into an indexed entry takes a little work, takes a little processing.

**[00:44:09] JW**: Takes a little work. I think the query side is in many ways more interesting to me. When you think of things that would like be like the end of Slack, the death of Slack, that we

were extra special paranoid about, Slack search is kind of like a Texas hold 'em. There are messages that you can see and only you can see, and there are messages that are public, that everyone can see to a given team, right? If we ever, ever return messages to people or files to people that they were not allowed to see, that would be an absolute catastrophe.

Whether it was from some other team or from other people in your team, whatever, like that kind of stuff, where at least, again, in my time, triple levels of redundant checking to verify that you could in fact see the message that you are allowed to see and to the extent the like Slack search can be a little slow. That's the primary reason why. Because we are utterly paranoid about the security and making sure that you are only seeing things that you are allowed to see, again, with the weird edge cases and all that kind of stuff. Generally speaking, that's sort of like the primary goal.

In that sense, it is also a little different than Elasticsearch. It is like a unique search problem, and I want to tie this back to what I was saying about choosing boring technology, because when we're building the Solar Cloud cluster and we were doing all of the stuff, we were really trying to do things by the book. There's a lot of infrastructure information expertise out there about Solar Cloud, and we were trying to use all of it, especially like the replication logic and stuff like that that I talked about in the Activate talk, and we just couldn't get it to work.

We just absolutely could not get sort of classic Solar Cloud replication in any modality including – I mean, we eventually like – We were running against master. We were forking master and adding interim patches to fix bugs we were running into. We just simply could not get it work. We could not get the cluster to be stable at all. Months and months and months of beating our head against this, trying to figure out are we stupid? Is there a chapter in the book we haven't read? What is going on here?

Finally, finally, through just kind of a fortuitous, just like the wonderful thing about living in San Francisco, is everyone knows everybody. It's a village, right? We reached out, back channeled some friends of ours with Salesforce and Apple and really basically what the fuck. What are we doing wrong? Bless their hearts, they just leveled with us and we're like, "Yeah, that stuff doesn't work." "At the scale you guys are running?" "It just doesn't work."

**[00:46:28] JM**: Oh no! That's not the answer you want to hear.

**[00:46:30] JW**: It's not. It's not. They're like, "We don't run that way. We can't run that way. It doesn't work for us." That was an incredible moment, because we then, a team of I guess like five or six of us at that point were like, "Okay. We need to invent a replication and redundancy strategy right now while Slack searches basically burning down all around us." Query latencies were off the charts. It is the single most expensive piece of our infrastructure. We spent more money on it than we do on the MySQL layer, than we do on the application tier.

**[00:47:02] JM**: It's what people pay for.

**[00:47:04] JW**: It's what people pay. Absolutely, the whole thing is burning around. So we need to invent it. Jeff, we did. We did. We invented it and it took about a week or so to invent it, and I think maybe a month or so to implement it. Then we got it up and spun them. This is like early January 2018 we're doing this work, and we launched on March 1st, 2018.

**[00:47:22] JM**: There was no one weird trick. It was just a bunch of grindy-grinding to get there, or was there a one weird trick?

**[00:47:29] JW**: Basically, we turned – I think the short hand I would say, we turned what we thought of as our emergency backup strategy into the way, would be the way I would say it, the way I would describe it. So we did a couple of things. One is we implemented – The old HDFS [inaudible 00:47:46], right? There're three copies of every block. The idea of HDFS or S3 or any of these systems is you want to have reliable data availability in the presence of like unreliable hardware. The only way to really do that is to like create multiple copies of everything. That's how these systems work.

It's actually how Slack search works as well. There isn't one copy of the message index. There're three copies of every message in the index. We have redundancy like in the kind of HDFS sense. That is expensive, but it turns out to be less expensive than running Slack search the old way, like to the tune of millions of dollars.

So we create – Like basically if a shard Slack Solar Cloud cluster dies, we know what sort of index segments, what are called segments, like the data files, were on that one. There are other copies of them elsewhere. We spin up a new cluster. We copy over the segments. We kind of like replay things basically to get them caught up to like the latest and greatest. If someone adds in an emoji reaction, that updates the document.

If someone deletes a document, that's obviously a thing, all that kind of stuff. We get them caught up we reintroduce them to the index. So there's generally like these little windows of time. Essentially, at any given time, Slack search, Solar Cloud cluster have thousands of node in it. There's one that's down always, but it doesn't matter, because everything is replicated. All the documents are available all the time in this like unreliable system. We basically implemented the HDFS trick ourselves on top of Solar Cloud to solve this problem. That would be the way that I would describe it. Yeah.

**[00:49:15] JM**: I'm just having trouble understanding why the – That's like a reliability or a durability problem.

**[00:49:24] JW**: Yeah.

**[00:49:24] JM**: I guess I'm having trouble understanding what was the bottleneck that that solved.

**[00:49:31] JW**: So the way that Solar Cloud reliability durability works would be the way that like sort of classic master – I don't like to say it. Master read replica. I don't want to say this. Master read replica is sort of MySQL application works. That's the way it sort of classically works, is Solar Cloud itself will – You write a record to a Solar Cloud node and it will replicate that record for you and the same way that like a MySQL database will replicate a row to a recopy. If that MySQL node dies or that Solar node dies, no big deal. There's a read replica ready to go to start serving stuff immediately. That's the way like a MySQL replica.

Solar uses a very similar kind of strategy, like a database replication strategy. Basically it's like the replication is in Solar Cloud. We don't do that. We do the replication ourselves to the

application layer. Solar Cloud is not allowed to do any replication. We manually, again, behind-the-scenes –

**[00:50:24] JM**: And you do it twice.

**[00:50:25] JW**: Three times replicate, yeah, every message into the index.

**[00:50:29] JM**: Sorry. So Solar replicates once. You replicate twice.

**[00:50:33] JW**: We do.

**[00:50:33] JM**: Okay.

**[00:50:35] JW**: I mean, Solar can replicate an arbitrary number of times in the same way that MySQL can have arbitrary number of read replicas. It's the same idea.

**[00:50:41] JM**: But you needed to write your own for some reason?

**[00:50:42] JW**: We did, yeah.

**[00:50:44] JM**: What was the problem with Solar Cloud's implement?

**[00:50:46] JW**: It's can't handle the write volume that we're throwing at it.

**[00:50:49] JM**: Oh! So you had to buffer it or something?

**[00:50:51] JW**: Oh! I mean, no. It can handle the write volume fine. You just need to shard it out kind of like broadly enough.

**[00:50:57] JM**: Oh, okay! You parallelized the indexing.

**[00:51:00] JW**: Well, yeah. I mean, obviously, you parallelize indexing. What I mean is we parallelize the replication. I have live like X-nodes necessary to serve my query volume, right? If

I'm going to have one replica, I'm going to have 2X those number of nodes. Does that make sense? Master serving queries, replica. Taking writes serving queries, replicas on the backend.

**[00:51:19] JM**: Yeah.

**[00:51:20] JW**: Instead of letting Solar Cloud handle that, that 2X replication, we just take the 2X nodes and you shard things out that much more. Does that make sense? Just like scale things out this way, and when we do the replication ourselves of the application layer. The application now has to do three writes, and of course there are like transactional sort of problems with that too. What happens if the process dies while it was doing the writes and blah-blah-blah-blah-blah? There's all these stuff that are going to happen.

You create a different set of problems for yourself, but they're like solvable, tractable problems in a way that like reinventing Solar Cloud's entire replication strategy from scratch in three weeks was going to be like tractable.

**[00:51:59] JM**: What problem did that solve?

**[00:52:01] JW**: Availability of the documents when a node goes down.

**[00:52:04] JM**: Okay.

**[00:52:05] JW**: So if a node dies, I want to make sure that I can still query the documents that were on that node. No node. Part of the induction like disappear just because some node dies.

**[00:52:16] JM**: Okay. In the solar version, you would've just been blocking too many times because there would be no node that was replicated or you literally would have like events where you would lose data –

**[00:52:30] JW**: Yeah. You definitely don't want to lose – I mean, losing data was the primary fear.

**[00:52:33] JM**: You would actually lose data.

**[00:52:35] JW**: I mean, no instance that like the message would still be persisted to MySQL and could be recovered later. Again, multiple layers of redundancy in every way, but the message would not be queriable for some period of time, like on the order of – It could be days. It could be hours. It could be that kind of stuff. The message would be unavailable for query. You could not find it, which again it's tricky with this stuff, but it's like if businesses, if it's like I need to find a receipt, I need to find an invoice, it's a big deal. I need to be able to find it right now. Yeah, anyway.

**[00:53:03] JM**: That's a problem that Apple or Salesforce would have encountered if they had the write frequency.

**[00:53:13] JW**: And they do. They do. I would say it's not even close. Apple and Salesforce are like orders of magnitude larger than Slack. Apple is like all of Apple mail is backed by Solar Cloud. Effectively, Salesforce is essentially like – I don't want to offend anybody. Salesforce is in my understanding a thin veneer on front of like a custom Solar instance. Search is, it's a massive interface for Salesforce. A lot of discovery navigation happens via search, and like every object in Salesforce is available for query in Solar and kind of has to be. They are by far the heavy duty. Yeah, absolutely, Solar users.

**[00:53:49] JM**: Why wasn't that bottleneck an issue for them?

**[00:53:52] JW**: It was.

**[00:53:53] JM**: Oh!

**[00:53:53] JW**: Yeah.

**[00:53:54] JM**: So when you went to them and said, "Hey, this doesn't work." They literally said, "Yeah, we have noticed that and we are suffering from it."

**[00:54:00] JW**: I would say we've worked around it I think would be what they would say. We have found ways to, yeah, deal with that constraint and ways that were appropriate for our

individual use case and our needs. We recommend that you use Slack. Also find a way to work around it that is appropriate for your use case needs. Basically, if anything, they were kind enough to tell us to like stop banging our heads against the wall, which in a lot of ways is the – I mean, it's kind of like, it's the – Just knowing that a problem is solvable is in a lot of ways all you need to solve it, right?

**[00:54:32] JM**: Well, and also that there's no easy solution. There's no off-the-shelf thing that's going to fix this problem. You're going to have to work through it yourself.

**[00:54:38] JW**: Exactly. I feel at some companies there is a strong tendency to like never run on this problem, because like they love reinventing the wheel or they're Google and they love vulcanizing their own rubber, like all that kind of stuff, right? They'll just reinvent the wheel. We vulcanizing our own rubber. That is the Google motto. Companies love this. A lot of engineers love it. I get. It's super fun.

At Slack, that was very much not our mentality and it was so much not our mentality that it took us a really long time. In fact, needed like a swift kick in the head basically to say, "Hey, Slack. You need to reinvent the wheel. You need in invent your own wheel. There isn't a wheel you can go buy. You have to go create it."

**[00:55:15] JM**: You said this actually impacted the client logic.

**[00:55:19] JW**: It didn't impact the client logic.

**[00:55:20] JM**: It did not impact the client logic.

**[00:55:21] JW**: Yeah, the virtue of this system by and large was this like we abstracted that stuff away through a proxy layer, so like the client didn't have to care.

**[00:55:28] JM**: Okay. That's great.

[SPONSOR MESSAGE]

**[00:55:38] JM**: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

**[00:57:28] JM**: Did you have caching layers you had to create on top of that search system or was there a best practice for dealing with the caching stuff?

**[00:57:38] JW**: Do be honest with you, no. Slack search doesn't cache I guess would be like the TLDR. Slack search doesn't cache. Beyond like the very minimal sort of like the OS doing SSD level caching for segments that get hit very often, there are teams that search a lot more than others and you can use this kind of – Again, you can use off-the-shelf like Solar Cloud caching. We can use our off-the-shelf kind of like message caching infrastructure for like pulling fully kind of hydrated messages from MySQL and sticking them in a cache. All that stuff was not, but we

experimented with a number of different kind of caching approaches. Generally speaking, beyond the fact that knowing that some teams query a lot more than others. Basically, the TLDR of this was that the thing we got from building the index offline via MapReduce was fundamentally the ability to restructure the index so that things would cache better. Putting all of the documents from the same team together in this sort of historical index was the single biggest performance improvement we got just by a wide margin.

Again, when we're doing writes, the messages were coming in from every team. They're coming in so fast. You just stick them wherever you can as fast as you can. But when you're doing the historical index, you actually have the opportunity to restructure things to make the reads vastly more efficient and vastly more cache friendly. That was massive. That was like to the tune of like 300, 400 milliseconds off of P95. That was a really big one. Yeah. But no additional custom caching. It was just being smart about how do you structure the index. Yeah.

**[00:59:08] JM**: Taking into your context your experience at Google and also at Cloudera. You were at both of those companies for four years respectively. You've been at other companies. Then your four years at Slack. What is distinctive about the Slack culture both from a product perspective, from a long-term perspective, and from an engineering perspective? What makes the company distinct culture-wise?

**[00:59:33] JW**: Slack is by far the most product-oriented company I have ever worked at. I have generally worked at engineering-driven, engineering led companies. Google is engineering-driven to a fault. Cloudera, I don't actually know what Cloudera is anymore. But when Cloudera was there, Cloudera was very engineering-driven very much so. Slack is not. Slack as product-driven in a way that was good with jarring for me. I was very surprised. It was very different than like the way I worked and stuff like that.

**[01:00:08] JM**: Product-driven meaning like top down. We're going to design a product then we're going to engineer around that.

**[01:00:12] JW**: Yeah, very much so. Very much so. Yeah. In a way that like Apple is product-driven, and the way like lots of very successful companies are product-driven. Again, coming from like my limited experience and limited perspective, I assume that all successful companies

were fundamentally engineering-driven companies. That's not Slack. It's just not. It's just not who they are, and it was painful, but again educational I think to work in a product-driven environment, where just really like design and product are really first-class citizens and not – Again, I can't speak for Google now, but not kind of like afterthoughts in the way that I felt like they were in a lot of ways at Google. Not afterthoughts. It's not the right way, but like fundamentally it was an engineering – Yeah.

**[01:00:55] JM**: Well, they have so much money there. They're now a product company and an engineering company.

**[01:00:58] JW**: Yeah. I mean, absolutely, and a design company, and they can absolutely throw enough money at any problem they want to be anything they want. No question about it. Whatever version of a company you like working at, Google's got something for you, I would say probably speaking. Slack is not that. Slack is a product-driven company.

Slack was by far the nicest company I have ever worked at. People were genuinely kind. By far, I think outside of the few people I know who got fired for bad behavior, I was the worst person at Slack, really. I was the least biggest asshole, least kind person at Slack. I generally think of myself as a kind of person which I think reflects in the amount of like obviously self-delusion and also the fact I'm probably like not that bad of a person in the grand scheme of things, but I was by far the worst person at Slack. Everyone there is very, very kind, very, very nice.

**[01:01:47] JM**: And a kind culture, the 9 to 5 culture.

**[01:01:49] JW**: The 9 to 5 culture is a real thing.

**[01:01:51] JM**: I love that.

**[01:01:51] JW**: I did too. I did too. I have a four-year-old. He was born a month before I started working in Slack, which again for you listeners, starting a job, a new job like one month after you have a child is a terrible idea. You should absolutely not do that. But, yes, 9 to 5. I get put my son to bed every night. I got to make him breakfast every morning. I'd walk him to preschool, all that kind of stuff. It's great. Absolutely fantastic. Yeah, I love that.

**[01:02:19] JM**: Let's take a step back and think about things in terms of the “broader industry”. You alluded to this earlier that we have in 2019 this robust buffet of different technologies we can work with, whether they're coming from cloud providers or companies that are very narrowly focused on some problem domain, like Fivetran, or Snowflake, or Databricks, or whatever, and it's a great time to be building a company, because really if you are a “product person”, if you have an idea for a product and you know something about software engineering, you can build it. It's just not that hard and it's only getting easier.

What are the shortcomings of modern data warehousing tools and data infrastructure tools?

**[01:03:11] JW**: That is a great, great question. What are the shortcomings of them? I think I had an answer for a long time. This is based on my experience at Slack. The greatest source of irritation for us was having to use Hive or Spark, SparksSQL, whatever, for ETL or heavy duty kind of intensive machine learning, and Presto for interactive query. That Hive and Spark, we just were not really great as good as Presto was and interactive, and Presto really wasn't as good at ETL sort of stuff as Hive was.

The fact that the query languages across the three are not close, but they’re not exactly compatible with one another was like the single greatest source of frustration. There was not one system that can handle all of our ETL-ish SQL and all of our interactive SQL on top of what was to be fair, essentially, an infinitely scalable data lake-ish thing built on S3 with one centralized hive meta-store.

I think Snowflake has done a phenomenally good job of largely solving this problem. I've been tremendously impressed with what Snowflake has built to be able to handle both of these things. I don’t actually know how they do it. I kind of am tempted to go work there just so I could like find out, or maybe I could just swing by and they would just – Maybe I could just ask them and they would tell me.

**[01:04:31] JM**: We did an episode on it.

**[01:04:33] JW**: I’ll just have to listen to episode then. That’d be fantastic.

**[01:04:35] JM**: It was interesting.

**[01:04:36] JW**: They can do both in one query language, and that is an amazing superpower. That just removes so much friction. It makes it so much easier to move back and forth from exploration to productionalization, all that kind of stuff. That's huge. That is absolutely phenomenal. Yeah, I'm a big fan of that and that's like phenomenally exciting to me.

I have to give kind of a long pontificating answer to this question, and even I find it a little tedious. Just edit out as much of it as you can. Analytics value chain. Okay? There's ingest, I need to get data into my analytical system. I need to store it. I need to do computation on it and I need to visualize, process the results in some way. There are companies that exist at like every sort of layer of this stack, right? There's like on the ingest side, you can get stream sets. Confluent does a lot of this stuff. There're a lot of companies, like classic Informatica, like classic kind of ETL like loading, the loading aspect of getting data around.

For storage, S3. Obviously, lots recommend it at Red Shift, if you're so inclined, blah-blah-blah. I mean, Snowflake I basically think it was S3-based storage more or less. For compute, Spark, Snowflake, Presto, Hive, a bunch of stuff I can't even think of, like all that kind of stuff. Then on the sort of visualization, like do stuff with it. I mean, everything. Everything, Tableau, Moe, Periscope. I don't know all the full set of that, Jupyter Notebooks. Then you have like solution providers that kind of provide any level of abstraction over this value chain you want. So you can buy like Sisense and they'll just do everything. You can do BigQuery, and they'll do storage and compute, but not visualization. Whatever sort of subset of this you want to have, you can have, and that's amazing. That's also like deeply confusing and scary. I think it's like overwhelming broadly speaking.

For a long time – I will still say to a large extent, the vast – Clayton Christiansen – I wear a hoodie and I dress like this, but I'm basically like an MBA student in disguise. Clayton Christiansen wrote the *Innovator's Dilemma*, like *Innovator's Solution* and all that kind of stuff and he has this great thing. I'm also a big [inaudible 01:06:39] guy like Ben Thompson.

**[01:06:40] JM**: I was just thinking, you look a little bit like Ben Thompson.

**[01:06:42] JW**: Do I look like –

**[01:06:43] JM**: Not in a bad way.

**[01:06:44] JW**: My wife is from Taiwan too. I should go move over and hang out with him. I don't know. Honestly, we visit relatives there sometimes and I'm like always kind of like vaguely tempted to see, like send them an email and say, "If you'd–" See if you want to hang out with me and maybe like getaway for me stupid dirty software engineer. But anyway – Yeah, the law of conservation of attractive profits, modularization and integration across the value chain, like that kind of stuff.

I am, as you would imagine, after like hearing me pontificate about the history of logging at Google, I am a student of history in a lot of different things. In the analytics value chain, for a really, really long time, all of the money was really at that storage compute interface. What I mean by that is like Teradata, like going back to the 90s when Teradata was dominant and it was all about like Teradata. Teradata built the very best system and they built the very best system because they integrated the ever-living fuck out of storage and compute. They wrote custom disk drivers, right? They use custom hardware. Netezza operated in the same model. Tight, tight integration between storage of data and query over data to get the absolute best performance you could over large volumes of data. It's amazing.

Obviously, I worked at Cloudera. Cloudera involved like kind of the commoditization of this stuff in certain ways and like Teradata was my great enemy for like four years. I guess Hortonworks was also tediously my great enemy for a number of years. But leaving that aside, kind of breaking down this like very tight storage compute integration to a Hadoop world where, yes, there were storate compute integration, but it was loosely coupled and you could sort of more – You had more flexibility and more control over it and all that kind of stuff.

When I got to Slack, we had bought in pretty hard to the Netflix style system where like S3 is the source of truth, because of course the problem with Hadoop and Teradata and Netezza and all these systems is they're great, including actually Red Shift now that I think about it. They're

great until you exceed the storage capacity of the system and you have to upgrade to the next biggest Netezza instance, the next biggest Teradata instance.

Red Shift is great when you have a petabyte of data and an absolute nightmare when you have 2 petabytes of data. It just falls off a cliff and you can ask anybody. That's why they have Red Shift Spectrum now to like help alleviate this problem.

Anyway. So we build everything on S3, and when you're building on S3 – S3, obviously, although I think they've started adding like effectively query capability to S3 overtime, S3 is really just storage. It's dumb storage. It's storage like disaggregated, modularized if you will, across the query interface. That's why at Slack we could use Hive and we could use Spark and we could use Presto and everyone could write data to S3 and everyone else could query it, and that was like super cool.

We had modularity and flexibility across these interfaces, and I kind of thought that was like the future, but then Snowflake came along. I think the thing Snowflake did really well is they built a compute engine that is optimized and worn and designed for S3 in a way that Spark and Presto and Hive are not. They've made adaptations to deal with S3 and object storage, but fundamentally they were born of HDFS and they all were and they're still largely run in Facebook and so on and so forth on top of HDFS or, again, derivations of HDFS.

Snowflake built a system that was born in the cloud or at least like really close to it and showed that if you do that, you can save just a tremendous amount of money, like an enormous quantity of EC2 cost, because you can scale this thing up and down incredibly flexibly. A few folks tried to do this. [inaudible 01:10:10] tried to do this. A lot of people tried to do this. Snowflake just did a better. I say that like success in my experience comes to people who get there first with the right solution. They got there first with the right solution. A lot of people can get there first, but like not get it right. A lot of people can get the right solution much later in life as Google has sadly learned a few times now. You get there first with the right solution and you win. That I think is like the most compelling aspects of Snowflake for me, like solving that problem.

I wonder how long this goes on though, I guess. I feel like there's sort of still – And maybe this is just wishful thinking on my part. I feel like there is still sort of an inexorable force that is pushing

like towards modularization of storage and compute where these two things do not actually have to be super tightly integrated together. Part of this is like the pain and suffering that I have gone through dealing with like Salesforce integrations or getting data out of whatever, whatever vendors that my marketing team decided to use and into the data warehouse and stuff like that. It was still a lot of tedium and work that was kind of like unnecessary in order to bring it into kind of like either the Parque, Hadoop, Hive ecosystem we created or into like the snowflake ecosystem. It's still like that ingest piece is still a lot of work.

When you have things like Mulesoft, which come along and say, “He, let’s just use APIs for data integration. Computers are good. We don't actually need that much data from Salesforce most of the time. We don’t need to pull over everything. We can just pull over the little bit we need and use it as we need it, right?”

Which again is back to the idea of decoupling storage and compute, that like storage can be in Salesforce, and it could be in Marketo, and it can be in S3, and it can be in wherever, and we can throw a query interface on top of it that can talk to all of those different things. That for me is like the most interesting kind of question for the next like five years. What happens to that storage computer interface? Because, honestly, that's where all the money is. So if you want – Tableau got acquired for whatever. Looker got acquired for whatever. That's fantastic. That’s great. I have no doubt stream sets will do really. But, fundamentally, the money in this business is at that storage compute interface and whoever like wins there, wins like the future of how we do things.

If it’s storage compute super tightly coupled together, you got to bet on Snowflake. If it's the sort of disaggregated decoupled system where like anything can talk with anything else, then I think you can think Presto, Starburst, Spark and Databricks, so on and so forth and maybe other people who haven’t been born yet and will create a solution that is born for that world could potentially win.

**[01:12:39] JM**: Right.

**[01:12:40] JW**: That's what's interesting to me.

**[01:12:43] JM**: Yeah.

**[01:12:43] JW**: I sound like a venture capitalist. Don't I?

**[01:12:45] JM**: Yeah, a little bit.

**[01:12:46] JW**: Ouch.

**[01:12:46] JM**: But this is the heart of what – I mean, we've done a lot shows about "data platform", and that's kind of what it is. For a lot of people, the more people I talk to, the pivotal decision does seem to be are you Snowflake or are you Spark disaggregated?

**[01:13:05] JW**: Yeah, that's right. Federation – To me, it's like federated independent source and compute versus tightly coupled, tightly integrated. That is the choice. The fact that you even get a choice is kind of exciting. Makes a very, very interesting time.

**[01:13:18] JM**: Okay. Well, a lot of stuff we didn't get to, but beginning to wind down.

**[01:13:23] JW**: Sure, man.

**[01:13:25] JM**: Early days of Facebook or early days of Google –

**[01:13:29] JW**: I wasn't there. Just to be clear. I'm a historian. I didn't live it.

**[01:13:33] JM**: I know you weren't there. But early days of Facebook or Google, nobody would've anticipated what these behemoths became. I mean, there was something in probably a kernel of what they eventually became and the vision that the founders laid out or whatever.

**[01:13:46] JW**: I don't think that's true.

**[01:13:48] JM**: Probably not.

**[01:13:48] JW**: Probably not.

**[01:13:49] JM**: Probably not.

**[01:13:49] JW**: No. No way. Absolutely not.

**[01:13:49] JM**: It's more like they eventually got infinite money and they were capable of dreaming up something to do with that infinite money.

**[01:13:56] JW**: They were.

**[01:13:57] JM**: Okay. Let's say Slack gets infinite money. What does Slack become?

**[01:14:01] JW**: Oh! Interesting. What does Slack –

**[01:14:02] JM**: What's the 10-year, 20-year thing that it morphs into?

**[01:14:07] JW**: I think that's not quite right. I guess what I want to clarify is like you don't decide what to become when you get infinite – I don't know. I'm sure you do have to decide what to become when you get infinite money.

**[01:14:18] JM**: The more clear question I'm looking for is what does Slack look like in 10 years? What's the vision for the company?

**[01:14:23] JW**: Oh! I have no idea, probably speaking. I mean, at a product-driven company, I don't think you can ask an engineer like what the vision is and get like probably like the best right answer not because it hasn't been explained to me many times, but because I am just like fairly dense and I just don't like totally get it.

I can tell you what vision I had for Slack, but it's very much like an engineer-driven vision. I don't think it's like the vision-vision I would say. I don't think I speak authoritatively, especially since, again, I don't work there anymore. I don't need Slack stock. I don't have any stake in the company's future.

**[01:14:57] JM**: Okay. All right. All right. Different question?

**[01:14:59] JW**: If you like.

**[01:15:01] JM**: Tell me what you think the CEO wants the company to become long-term?

**[01:15:05] JW**: Man, I really – Again, I can tell you what I think the company should become, but I –

**[01:15:09] JM**: Okay. Sure. Sure. Tell me that. Very condensed.

**[01:15:13] JW**: Slack is remarkably a successful company. It's an incredibly successful company, even right now at 10 billion or whatever it's worth. That is outrageous. But if slack in 5 or 10 years is only a $10 billion company, it will be like one of the saddest, most disappointing episode of my life, right? If it's a $20 billion, if it's anything less than 100+ billion dollar company but it's not one of those companies, it is an absolute disappointment at least for me, at least for me, in spite of all of its success, in spite of everything.

How does it become that? It would need to do two things I think. One of them is sort of predicated on the other but doing either one of them would be sort of sufficient I think for being like one of the most valuable companies in the world. One is that Slack certainly has the potential to become the modern, I don't know if you call it the ERP system, but like the modern sort of fundamental integration point of all business processes. That every single system in the way that like if you ask Confluent, what is their vision for Kafka, right? Kafka is at the center of all things. Kafka is like every sort of data process at the system flows through Kafka in some way.

I think for technical events, I think that could very much be true and you can build the – I think they would beat the crap out of me if they heard me call it an enterprise data bus, but whatever that sort of – The circulatory system. Whatever you want to call it, like the nervous system, blah-blah-blah-blah-blah. Slack could become the human version of that.

The Kafka thing is great, but it's not really for humans. It's for computers. Slack could become that sort of central nervous system hub that like all consequential events in a company flow through, like in a way that like if you run your company on SAP and you're doing on your manufacturing widgets whatever, fundamentally, like your company runs on SAP in that kind of way. Everything runs through SAP systems or the Oracle systems or whatever, whatever, right?

In the like cloud world we live in now where you have all these different vendors and all these different systems and all these different people, Slack is the thing you could plug everything into. That is what it is designed to be. You could pipe every single event, and this is like largely speaking how Slack itself runs, right? Slack runs on Slack. Slack pipes everything into Slack. Slack does everything on Slack almost to a fault. Even when the product isn't quite ready for some of those things that Slack wants to do, but that's how they find out, right?

Being that hub is an incredibly valuable place to be, because it would enable, I think, you to build – And I just kind of want to punch myself as I say this, like it would enable you to build like the AI sort of machine learning-driven systems that would help people run their businesses better, and you could be like really like the global brains of the operation in some nontrivial sense.

If you expand that vision from a single company in a way that I think Slack is doing right now with their efforts around shared channels to build this web of companies, because no company is an island, right? You have suppliers, you customers, you have Michael Porter's Five Forces every which way. Do my MBA thing again. If you can tie all of these companies together into one sort of global network, if you basically build the equivalent of like Facebook but for businesses where everyone is connected to everyone else, anyone is reachable through anyone else, be it like the network of shared channels.

Any business process can reach any other business process anywhere. You are the most valuable company in the world without a question. I don’t think it's close. I mean, to be honest with you. You're more valuable than Google and Facebook together. That's the goal. That's what – If it's not, it really should be.

**[01:18:45] JM**: Well, I remember, I saw some tweet or post or something about the shared channels and how difficult that was to implement.

**[01:18:52] JW**: It's incredibly difficult to implement. Yes, absolutely. Very, very hard to do.

**[01:18:56] JM**: Final question. As you mentioned, you have done these separate four-year stents. You've done some other stuff.

**[01:19:03] JW**: I have? That's pretty much it. Like I said, I did do some other stuff once. Who can remember? Yeah.

**[01:19:08] JM**: You said you've been in the working world for like 20 years.

**[01:19:10] JW**: Yeah, almost 20 years.

**[01:19:12] JM**: Any distinctive perspective on how the nature of work is changing? This idea where most of us are going to offices, the way the companies are built. I mean, obviously we can have the typical conversations about going towards remote work or whatever.

**[01:19:32] JW**: Remote work. Yeah, exactly.

**[01:19:34] JM**: Any just subtle or distinctive thoughts that I might not hear from someone else?

**[01:19:39] JW**: I don't really have any. I mean, it's not just something I give a lot of thought to.

**[01:19:42] JM**: Okay.

**[01:19:42] JW**: I'm just being honest with you. Yeah.

**[01:19:43] JM**: Okay.

**[01:19:43] JW**: I do not feel like I have anything meaningful to add.

**[01:19:45] JM**: All right.

**[01:19:46] JW**: Sorry.

**[01:19:47] JM**: That's fine.

**[01:19:47] JW**: Yeah.

**[01:19:49] JM**: Closing thoughts?

**[01:19:49] JW**: I feel like in many ways me saying no, I don't have any meaningful to add is a pretty good metaphor for this entire talk. I would probably just leave it at that. Yeah.

**[01:19:57] JM**: All right. Fair enough. Fair enough.

Josh Wills, thanks for coming on the show. It's been really fun talking to you.

**[01:20:02] JW**: Thanks so much for having me. Good stuff, man. Thanks a lot.

**[01:20:03] JM**: Yeah, great conversation.

[END OF INTERVIEW]

**[01:20:14] JM**: As a programmer, you think an object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud area. Millions of developers use MongoDB to power the world's most innovative products and services, from crypto currency, to online gaming, IoT and more. Try Mongo DB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[END]