EPISODE 482

[INTRODUCTION]

[0:00:00.6] JM: Google Docs is used by millions of people to collaborate on documents together. With today's technology, you could spend a weekend coding and build a basic version of a collaborative text editor, but in 2004, it was not so easy. In 2004, Steve Newman, built a product called Writely, which allowed users to collaborate on documents together.

Initially, Writely was hosted on a single server that Steve managed himself. All of the reads and the writes to the documents went through a single server. Writely rapidly grew in popularity and Steve went through a crash course in distributed systems as he tried to keep up with the growing user base.

In 2006, Writely was acquired by Google and Steve spent his next four years turning Writely into Google docs. Eventually, Steve moved on to other projects within Google; Cosmo and Megastore Replication. When Steve left the company in 2010, he took with him the lessons of logging and monitoring that keep Google's infrastructure observable.

Large organizations have terabytes of log data to manage. This data streams off of the servers that are running our applications. That log data gets processed in a metrics pipeline and turned into monitoring data. Monitoring data aggregates log data in a more presentable format. Most of the log messages that get created will never be seen with human eyes. These logs get aggregated into metrics and then compressed, and in many cases they eventually get thrown away.

Different companies have different sensitivity around their logs, so some companies may not garbage collect any of their logs. When a problem occurs in our infrastructure we need to be able to dig into our terabytes of log data and quickly find the root cause of a problem. If our log data is compressed and stored on disk, it will take longer to access, but if we keep all of our logs in memory, it would get really expensive.

To review, if I want to build a modern logging system from scratch today, I need to build metrics pipeline for converting log data into monitoring data. I need to build a complicated caching system, a way to store and compress logs, a query engine that knows how to ask questions to the log storage system, a user interface so I don't have to inspect these logs via the command line and the list of requirements goes on and on, which is why there is a huge industry around log management, and logging keeps evolving. One example we covered recently is distributed tracing, which is used to diagnose requests as they travel through multiple endpoints.

After Steve Newman left Google he started Scalyr, a product that allows developers to consume, store and query log messages. I was looking forward to talking to Steve about data engineering and the query engine that Scaly has architected. But we actually spent most of the conversation talking at the early days of Writely and how he scaled that and his time at Google, particularly the operational challenges of Google's infrastructure. So it was instructive about how Google's observability works.

Full disclosure, Scalyr is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:03:28.2] JM: Amazon Redshift powers the analytics of your business, and intermix.io powers the analytics of your Redshift. Your dashboards are loading slowly, your queries are getting stuck, your business intelligence tools are choking on data. The problem could be with how you are managing your Redshift cluster. Intermix.io gives you the tools that you need to analyze your Amazon Redshift performance and improve the toolchain of everyone downstream from your data warehouse.

The team at Intermix has seen so many red shift clusters that they are confident that they can solve whatever performance issues you are having. Go to intermix.io/sedaily to get a-30 day free trial of Intermix. Intermix.io gives you performance analytics for Amazon Redshift. Intermix collects all your Redshift logs and makes it easy to figure out what's wrong so that you can take action all in a nice intuitive dashboard.

The alternative is doing that yourself, running a bunch of scripts to get your diagnostic data and then figuring out how to visualize and manage it. What a nightmare and a waste of time.

Intermix is used by Postmates, Typeform, Udemy and other data teams who need insights into their Redshift cluster.

Go to intermix.io/sedaily to try out your free 30-day trial of Intermix and get your Redshift cluster under better analytics. Thanks to Intermix for being a new sponsor of Software Engineering Daily.

[INTERVIEW]

[0:05:13.3] JM: Steve Newman is the founder and EEO of Scalyr. Steve, welcome to Software Engineering Daily.

[0:05:18.2] SN: Thanks.

[0:05:18.7] JM: We're going to get into talking about Scalyr and modern log management and how high-volume log management works. I do want to start with a conversation about your company, Writely, which actually became Google Docs, and I use Google Docs on a regular basis. In fact I'm looking at a screen right now where I have 20 different Google Docs tabs open. So I'm definitely a power user of what your product became. I'd love to talk about the engineering challenges of building that, because back when you built it, I think like 2004, 2005, around that time, you did not have the tools that we have today to build distributed systems very easily. So what were some of the engineering challenges of building Writely in the early days?

[0:06:07.7] SN: Yeah, things were a little bit different than — We built Writely in 2005, which was especially on the client side, the browser side, was right at the edge when that was becoming possible. Gmail had just launched the year before with Rich Text editing in the compose window and that was actually one of the key inspirations for us.

A lot of the challenges were on the browser side and I can dive into that if you're interested, but JavaScript was much more primitive than everything about the browsers, was much more limited. The server-side, there are really kind of two stages to the story. When we originally launched Writely, we went from — One of my cofounders coming into the office and saying, "Hey, I've got an idea," and then basically describing Writely, from, "Hey, I got an idea to launch with about 100 days.” We put together just the minimum product we could as quickly as we could, and we launched it running on one server, one Windows server, ASP.net, some C# code we had thrown together and it kept all documents worldwide in memory on that one machine.

A few weeks later we got written up in TechCrunch and suddenly people started using it, and so scaling up that completely ad hoc intended as a prototype, Windows-based server. That was the first fit stage of the challenge. Then less than a year later when we were acquired by Google and suddenly had to face of a whole other several orders of magnitude of scale, getting ported under Google infrastructure and then writing that growth curve, there’s a whole, whole piles of war stories in all of that. May be single biggest challenge was actually immediately after we were acquired and porting a C#, ASP.net-based application over to Google friendly stack, which meant Java and Linux and using Bigtable instead of our homegrown ObjectStore and sitting behind Google frontends. There was not any part of the application that didn't change. Every box on the architecture chart, every line of code had to be rewritten in a new language and we had to do that before the — While we are doing that we weren't maintaining the old system. So we had to kind of build the new skyscraper before the old one collapsed under its own weight. That was one of the most intense things I’ve ever done. We did that in another — It was about 100 days.

[0:08:34.2] JM: I actually heard an interview recently with Paul Buchheit about building Gmail, and one of the things that he said was that JavaScript back then was really not built to do that level of intensity, like Rich Text editing in the browser where it’s going to automatically save on a regular — I'm sure it didn't automatically save on a regular basis back in the day. You probably did not have much fault tolerance or durability. I mean it sound like It was the same with Writely back in the day, because especially if you just have all of the documents on Writely just sitting on one box in memory, that is the opposite of durability.

[0:09:18.1] SN: That's right. I mean there was a copy on disk and we would update that copy. I don't even remember, but I'm sure we would update the copy on disc instantly, but it just had to fit in memory. The system wasn't designed to work — So every time the server would restart, it would load everything back from disk. Yeah, definitely working from the browser.

Writely, I don't remember either by Gmail, but yeah, probably it didn't save your draft until you click the save button. Writely did save every 20 or 30 seconds, and more often than that, if it saw someone else had the same document open, because of course that was what enabled us to then synchronize to the other browser and we didn't want much of a delay there. So the dumb way of doing that would be to copy your entire document from the browser up to the server every few seconds, which never mind what that would do to our servers. It would've been — People didn't have as much bandwidth back then. It would have been a challenge.

But just writing the most basic intelligence in JavaScript to decide, "Has this document changed? Which part of it changed?" We thought the dumbest thing we could do is just make a single loop through the document and look at every character to see if it's different. No fancy diff algorithms or anything, just loop from beginning, character zero to character N-1, and even that was too slow it turned out. That would take many seconds in JavaScript and lock the browser up. So we had to come up with the usual hacks.

I think one of things we did is a binary search calling substring on. So it did the whole document change, or it did half of the document change or it did a quarter of the document change. We did a binary search that way with substrings, because even just looking at each character once was too much. Yeah, I can only imagine what they had to do to make Gmail work even a year ahead of when we were doing Writely, but I'm sure there are a lot of hex like that.

[0:11:09.1] JM: The idea of collaborating on a document, and you have multiple people that are submitting changes to a document at any given time. Challenge similar to this was actually one of the — I took this distributed systems class in college and was one of the first projects that we did where basically what do you if people commit changes at the same time or rivaling times, and then there's some kind of latency in the system. Even though I wrote a change to a document five seconds before you did, then three seconds after you wrote my change — You wrote a change, but due to some latency, my change did not make it to the centralized server before yours. You've got these race conditions that could very quickly develop in a real-time collaborative system like that. Had you studied distributed systems in college? Were you familiar with the canonical kinds of race conditions that can develop in the kind of system?

[0:12:11.5] SN: I had a couple of classes about that. To some extent, I'm one of these self-taught people. I've been programming since I was eight and I kind of —

[0:12:22.5] JM: You got intuit, intuit those issues.

[0:12:24.3] JM: I remember — one project that I've done a long time ago, we developed a multiplayer video game on the Macintosh, a friend and I. One of the same people who's involved in Writely with me, Sam Schillace. Back in '91, '92 we had written this real-time 3D multiplayer tank battle game on the Macintosh and there is no race conditions, like I'm pushing the shoot button at the same time that you're pushing the dodge button and we're on different computers over the network. That was maybe my early trial by fire on that sort of thing. This sort of thing I'd seen before by the time we got to Writely.

The way we approached it actually was — And this is not the approach that's used today on Google Docs or Google sheets, but the way we originally built it, I actually drew — Kind of the inspiration we went to was source code control. We thought of it as a three-way merge problem. Internally we would maintain a revision history for every document. When you're editing — Basically, every time you hit the keyboard, you're making a branch and you're appending changes in your branch and someone else on their browser is appending changes to their branch. Then every time you synchronize back to the server, the server would merge those branches and if two different people have edited the same document, we would diff the documents and apply a three-way merge very much like what might happen in GitHub or something when you're merging branches.

It worked pretty well if you're editing different parts of the document and it would get interesting if you are on top of each other, and we had some heuristics for that. I think ultimately if you really stomped on, edited the same text, I think we would just throw both copies in. We weren't in a position to do, "So here's your version and here's the other person's version," because we weren't in a position to do anything more sophisticated.

What made this twice as hard was that the documents have style in addition to the text. So we have to think, "What if you changed a word and I made it bold." But what made it 10 times as hard was the documents we are working — We were thinking of this as HTML source code was the data model, and every browser had a slightly different way of doing that. So if you hit the return key in one browser, you might get a BR in another browser, you might get a P-tag. Then we would sink those back and forth, we'd send it from Netscape over to IE, and IE would rewrite all the piece as BR's or vice versa or something, and I don't mean to pick on IE, because all the browsers were doing things like that and they were all doing them differently. So every time we get the document back, it was completely different just because the browser had rewritten all the texts. That was, in truth, may be the single biggest issue that we wrestled with in the entire early history of Writely.

[SPONSOR MESSAGE]

[0:15:12.0] JM: You are building a cloud-native application and you need to pick a cloud service provider. Maybe you're just starting out with a new app, but you have dreams of scaling into the next giant unicorn. Maybe your business had been using on-premise servers and you want to start moving some of your infrastructure to a secure cloud provider that you can trust. Maybe you're already in the cloud, but you want to go multi-cloud for added resilience.

IBM Cloud gives you all the tools you need to build cloud-native applications. Use IBM Cloud Container Service to easily manage the deployment of your Docker containers. For serverless applications, use IBM Cloud Functions for low-cost, event-driven scalability. If you like to work with a fully-managed platform as a service, IBM Cloud Foundry gives you a cloud operating system to control your distributed application.

IBM Cloud is built on top of open-source tools and it integrates with all the third-party services that you need to build, deploy and manage your application. To start building with AI, IoT, data and mobile services today, go to softwareengineeringdaily.com/ibm and get started with countless tutorials and SDKs. You can start building apps for free and try numerous cloud services with no time restrictions. Try it out at softwareengineeringdaily.com/ibm.

Thanks again to IBM for being a new sponsor. We really appreciate it.

[INTERVIEW CONTINUED]

[0:16:49.0] JM: So these distributed systems challenges that we’re referring to here were a microcosmic example of some of the things that Google has ended up dealing with. For example, I did a couple of shows about Spanner, and I think Spanner is a database enabled by like atomic clock systems or just these crazy things that Google has built basically to solve those kinds of race conditions that can develop and other kinds of just distributed systems problems that you get when you're operating a global scale company that just runs off of distributed systems technologies.

When Google acquired Writely and it became Google Docs and you went through lots of production horror stories that I'm sure you could talk about and would be interested. We can fill the entire show with that. I want to move forward though, because we’ve got a lot of stuff to talk about.

A little bit about your time at Google, after the acquisition you worked on Google Docs for a while. I'm sure you're involved throughout your tenure there. We also worked on some big distributed systems, problems, or more generally worked on Cosmo and Megastore Replication. I don't know exactly what these were, but maybe you could talk in broad strokes about building these kinds of crazy infrastructure problems.

These have become almost pedestrian. We expect Google to be solving these problems at this point, but back in the day, I did show a while ago with a guy named John Looney, he was talking about doing these kinds of projects early days at Google and it was kind of groundbreaking and crazy.

[0:18:32.2] SN: Yeah, it was an interesting — It was a very interesting place to be. I learned a ton in the years I was at Google, but it was — We were acquired in 2006, and at that time Google already had very large and sophisticated infrastructure, everything from the physical hardware and virtualization. On top of that, to systems like map reduce and Bigtable and GFS and so for. No Spanner yet, but a lot of the things that we think about is being, mega scale Google engineering were already in place. But they were really, certainly at that point, still very optimized to support web search and some of the things in the ad business that had a similar flavor.

Search, for example, certainly then was basically stateless. When you interact with the search engine, you're not changing any data. You're just consuming the index. I think much less true now where it’s keeping your history and personalizing results a lot more than used to be the case then. Back then, they had 13 copies of the index all over the globe and they only needed 10 of them to function at any given time. If Atlanta goes down, fine, you’ll search out of Oregon. You won't know the difference.

That was fine for Web search. It's much less fine for a transactional user data application like Gmail or calendar or Google Docs where if you're editing your document and then something happens behind the scenes and suddenly your documents gone or you’re seeing the version from five minutes ago, that is not in any way okay. But the infrastructure was all designed to have massive scale, very cheap, no redundancy or minimal redundancy in everything from networking to power supply, planned maintenance all the time.

When you actually had transactional stateful systems where you couldn't just fail over, drop one data center and pick up the next one without advance notice, made it a lot more challenging, and there weren't any — When I arrived, there were not good systems in place to replicate data live across data centers. Gmail had something they had built that was being used by couple of other projects, but it was very raw, very hard to work with.

Bigtable was there and with a nice service, but only operated within a data center. So one of our big challenges was how do we keep Writely and then Google Docs running when one data center can go down and there's not a good way to replicate data to a second data center, and so that became the Megastore Replication project and that was myself. Basically, we cobbled together and a team from — I came off of Writely and other people came off of other projects with similar needs and we all sort of pulled together and said, “Let's build a system that can replicate data live and continuously across data centers so that we can give the uptime and reliability that we want to give to the user while working in the style of data center, cheap and not highly available. Let’s say cost-effective and not highly available that Google optimizes for,” or at least at that time. That was megastore replication, and there's been a nice paper that was written about it by some of the other people on the team.

In brief, it’s a paxos-based consensus algorithm for live synchronizing multiple Bigtable instances across data centers. So it's optimized for replicating across the globe or across the continent. So dealing with high latency and sometimes slightly flaky network between your replicas without letting that impact latency.

One of the key conundrums we wanted to solve was when you go, especially just to load your document, we want that to come back instantly so we don't want that have to consult multiple data centers. It will only just go to your nearest data center, get the document and give it back to right away and yet be certain that you'll get the latest data, but also be ready to tolerate that data center going down, which technically is a violation of the CAP theorem, and so we had to figure out in practice what's the best way to — You can sort of squeeze the CAP theorem down into less and less probable scenarios where you’ll actually trip over it.

So we went the route where it will never give you false data, but there are narrow cases where the system might become temporarily unavailable until a human operator intervenes. But working out all the details of that was one of the more interesting distributed systems challenges I've ever had to work on. I was lucky to work with a few very smart folks and we’ve kind of figured it out together.

[0:22:58.4] JM: Were you putting those systems into production or were you mostly synchronizing with people and doing white-boarding and thinking about the problems? Did you operationalize these things or were you more of an architect?

[0:23:10.5] SN: Yeah, I was involved end-to-end. There were three of us who started the project, and so we were brainstorming and white-boarding and going around other team who we thought might use it and kind of doing that early, what should we even be building and what are the requirements. All the way to writing the code, getting it deployed into production, keeping an eye on the pager.

Around the time that Megastore Replication went into production, I started in on this Cosmo project, which we can talk about. So I personally got a lot less involved around the time that we hit production, but it was the same team all the way through you. I don't think we had — The SRE was definitely around a Google than, but unless you are on one of the big moneymaking operations, like search or ads, it was very hard to get SRE support. At the beginning, we were doing that ourselves.

Again, personally, I wound up moving off that team around the time we hit production. So I was less involved, but otherwise, definitely the folks I've been working with, they were the ones carrying it to production.

[0:24:11.5] JM: This is a window into the world of the fact that the stuff that Google is adopting is typically 7 to 8 to 10 years ahead of the curve. Today, companies are building this SRE role into their engineering organizations, but it sounds like Google, even back then was starting to formalize this idea of the SRE. I want to start talking about that because I want to get into the company that you're building today, which is Scalyr.

In terms of operationalizing these difficult infrastructure challenges, do you feel like when you're looking back, was that a preview of the kinds of infrastructure challenges that people are dealing with today or maybe were they even harder, because today we just get to take advantage of cloud services and everything is kind of — A lot of the operations, you can just outsource. How did the challenges, the operational SRE challenges that you saw back then, how to those compare to the challenges of an average company today?

[0:25:19.4] SN: Yeah, it's a great question and I think you're exactly right that being at Google, really, with a preview of the future in this regard. Already, when we are arrived there in the acquisition in 2006, I don't think we were using the word cloud than, but it was deploying something in production at Google, there was a lot of analogy to building on AWS or Azure or Google cloud and so forth today, where there were all these services that were being provided for you internally and you're just a customer instead of building it yourself.

We got to appreciate a lot of the experience, a lot of the good and bad sides of that. The good side of course being here's all these things, services that are there for you. You don't need to build them. You don't need to deploy and maintain and monitor them specially this being Google. They're massively scalable. We're sitting on top of systems like Bigtable and sitting behind the same frontend traffic routing system that Google web search and ads use. Scale was no problem at all, but the challenge was things — It was a very noisy environment.

One story I like the tell, and I'll make it very brief here, is not long after we ported Writely across to Google infrastructure, we had a 20 minute outage, because it was halftime at the World Cup and everyone in Brazil — Brazil's team was playing, went under their social network to gossip about the game. At the time, if you were in Brazil, your social network was Orkut, which was a Google property, and they just melted the network in Orkut's data center, which also happened to be our data center.

[0:26:51.6] JM: Noisy neighbor.

[0:26:53.6] SN: Yeah, noisy neighbor. Certainly at that time, a lot of the Google services were not great about noisy neighbor control, and so actually that kind of noisy neighbor problem on the data center network, on disk I/O and the cluster that was sitting behind big table on pick your piece of infrastructure. At some point we had a noisy neighbor problem with it. That led to a lot of, “Okay. What’s going wrong now? Why is the site slow this afternoon? Why is this subsystem failing? Let's go investigate that.” We're doing investigation after investigation after investigation, and it’s that experience of trying to go from symptom to cause, digging into the logs and digging into the metrics and digging and all the other data that was being collected operationally. Indirectly, a lot of that wound up feeding into the ideas behind Scalyr. It was a fascinating education on kind of the good and the bad and the different ways of doing things.

One thing that I think was different about that environment than the cloud today is on the one hand the services weren't as polished as something you’d go by commercially from Amazon or whoever. But in the other hand they were a lot more transparent. So there might be more problems, but if the Bigtable service wasn't working, you can go — Within Google, you could go open up — You could go directly to their internal monitoring dashboards and see how much traffic is that service getting from everyone, not just from you, and how’s it performing and what does the CPU load look like on their servers. You could dive as deep as you wanted into the monitoring and the logging of all the services underneath you. Contrast with today where people talk about something like AWS where — I mean the truth is those services are generally very, very reliable, but when they are not, you can go look at the sea of green checkmarks on the status page and maybe there is a little I next to one of the green checkmarks, and that's the sum total of your visibility into what's going on underneath you.

[0:28:49.4] JM: Yeah, you get some partial failure on Redshift and it totally takes you down and you have no way to introspect into red shift. There's no debugger that allows you to step through what Redshifts is doing. It's just you've outsourced that.

[0:29:06.7] SN: Exactly. Even I'm having a problem with Redshift. Is that a system-wide Redshift issue? It's completely going off the rails. They obviously know if I'm complaining, I'm just distracting them, or is it a problem with Redshift, but it's narrow and maybe it's just me or a few other customers, and they may not know, or is it actually my problem and the reason Redshift isn't responding is I am sending the wrong API key or something. Is it on my end entirely? It's very hard to localize things like that now.

[0:29:39.0] JM: So I was at Amazon very briefly before I started this podcast. Just about eight months, but I was there long enough to see some production issues and see how logging works at one of these titanic companies, and it is a project. Because every service has to do logging, so if you're a company like a Google or an Amazon, there is going to be some platform engineering team that's like, "Okay. We are going to solve logging for the company," or me maybe you have multiple platform engineering teams with different solutions, but you don't want to have like the Google slides team have to roll their own logging system. You want some kind of standardized log or a semi-standardized logging system or a buffet of logging options. From looking at Scalyr, that seems to be kind of the approach. It's like, "Let's figure out how actually logging should work at scale and offer it as a service to people," and makes complete sense. What did you learn about logging specifically at Google?

[0:30:43.0] SN: A lot. It was actually interesting. One of these projects, I think it was the Cosmo project at Google. I talked about we have noisy neighbor issues and other issues. There are a lot of things to investigate. We had a meeting of the engineering team at one point to sort of compare, swap best practices, share tips on how different people would investigate things, and we made a list on the whiteboard of all of the different tools that we are using for working with logs and metrics and other kinds of visibility, data. The list, this is emblazoned in my memory, got to 17, 17 different tools all under the heading of gathered data or view data. Not to push, not to restart servers, not to take action, not to — Just to learn. It was because there were different kinds of data involved; logs, metrics, error reports, RPC traces, distributed traces, different ways of analyzing it, looking at one server at a time, looking at the global picture, different timescales, “Show me what's happening right now.” Real-time snapshot, or “Show me history over the last month.” All these different ways of slicing and dicing. So none of those tools could replace any of the others.

That was actually — The vent rate there was the original idea behind Scalyr, like create a single broad-spectrum tool where you can take all the different kinds of — Because learning 17 query languages and inking through, “All right, if I want to look at this data on this timescale, on this sort of spatial scale, or I think that tool might —” Just so much mental overhead and operational overhead and training overhead, juggling so many different tools.

The original idea was give you all your operational data in one place, sort of your classic engineer’s dream of, “Whatever the problem is, let's find the orthogonal solution and make everything orthogonal.” All the different data, all the different timescales, all the different analyses, let's do that in one place.

The other thing that I — It sort of took as a given is, “Whatever we do, it has to be fast,” because often when you're going to a tool like this, it’s because you've got a problem. It may be a really urgent problem like your site is down and you need to solve that now, and you're going to ask a lot of questions often, “All right. What's going on over serving errors?” “What kind of error?” “Which servers are having those errors?” You’re going to fire off often 30 or 40 questions. Boom! Boom! Boom! From symptoms such as; I can’t load the homepage, down to your root cause, “We messed something up in our database schema,” or whatever it's going to turn out to be.

When you need to do 40 things and you need to do them in a hurry, you want it to be fast, and that turned out to be what people really picked up on when we were showing around early prototypes, was just the speed of what we've done. So we’ve wound up focusing down a little bit from that original vision, just focusing on log management, which is often a speed challenge for people. We can talk about that a little bit more, but the original idea was that there're so many different aspects to visibility, so many different kinds of data that you’d want to look at, so many different ways you want to slice and dice them. One scenario is like grab the error logs and let me grab to see how often this error is happening, but it can get a lot more sophisticated, because often the nature of a problem can be unclear, like, “Oh, suddenly the database is slow.

Why is that?” A good guess is that we’re using it differently, where our queries got more expensive or we’re sending it more queries or something. Well, you look at the queries per second and that graph didn't really budge, but it could be that there's a little subcategory and they were very expensive queries and those went from 1/1000th of your mix to 1/100th of your mix, which you’re never going to see on the top level graph, but is crushing the server. Being able to really digging in —

Kind of getting back to your question about lessons from Google, one of them is you really need to be able to dig in and find those subtle patterns, and to do that you need a lot more than just, “Let me look at the top level metric, or let me search for the error message.” You really need to be able to slice and dice detailed log data to have any hope of tracking down a lot of the problems that come up. That’s kind of what we focused on.

[SPONSOR MESSAGE]

Learn about Truffle and download it directly from softwareengineeringdaily.com/consensys to get going on Ethereum development. If you want to hear a show about one of the topics that ConsenSys knows a lot about, send me a tweet @software_daily and tag @consensys, that's ConsenSys with a Y instead of a U, ConsensSys, with the topic that you would like to hear about. Tag both of us and let us know the topics that you're interested in hearing about. Thank you.

[INTERVIEW CONTINUED]

[0:36:54.3] JM: If I want to build a system that allows the engineers when there some kind of production issue and they want to dive into the logs and diagnose the issue. Yeah, I think that is the first consideration you want to take if you're building this logging system, because logs are great for solving problems. They're obviously great for having operational metrics and measuring KPIs and stuff to, but you need logs in order to diagnose a problem men.

A part of this is like the engineers — Regardless of whatever logging system you built, the engineers are going to have to do something to manage their logs correctly. They're going to have to be configuring some things correctly, because if you gather everything, if you gather everything and you just gather, you don't set a sample rate or anything. There's going to be so much information, and managing that information and putting it in a compressed state, then putting it on S3 or something like that. It can get very expensive. Anyway, I'm kind of rambling here.

Also, in a situation where you're going to diagnose a problem, especially with the certain cloud, big cloud systems today, you've got a queuing system in one place. You've got a database in one place. You've got a micro service somewhere or you've got a function as a service that is hosting your function. There' re all these different pieces and you almost need a snapshot of the entire system when an error occurs. So you've got this collation problem.

Anyway, what I'm trying to asymptote towards is the question of, "When you're trying to create a system that allows people to diagnose errors in an ad hoc fashion, when an error occurs, I want to be able to localize where that errors occurring. I want to be able to zoom in on the system that might have caused this exception when I'm dealing with this distributed infrastructure. What are some of the chokepoints where if I'm — I'm having trouble like articulating the right question. Yeah, the chokepoints in building a system that can intelligently manage all of that log data.

[0:39:21.0] SN: Yeah. Of course, it's always a trade-off. The more you can log, the more value you'll get out of it, but as with almost any engineering question, ultimately it's cost-benefit. Where do you get the most value for your log byte?

I think the two things that I think usually our best to focus. One is exactly what you identified, is chokepoints or kind of boundary — To start with, you've got a problem with your system. Often, it's an external symptom you're seeing. Things are loading slowly, your people are seeing error pages or something or maybe you're seeing something internal, like CPU usage is spiking on some of your servers or something, but it's a very sort of diffused external signal and you've got to narrow that down, “I know the site is slow, but what is that? The databases? That the Web servers? All right, if it’s in the web server, what about the web server?” You’ve got a narrow it down.

Where you want to focus a lot of your logging is on the communication between the boundaries of the different components in your system. So one nice thing — Happily, especially in modern systems, usually your system is divided into a lot of services. You've got maybe some kind of frontend web server tier and then application servers and database and maybe there’s a queuing system in there or an email system and some other component. You’ve got all these different components that are running on different machines. Many of them are actually just services that you're now renting instead of running yourself and they have very clear boundaries between their HTTP requests or RPCs or other kinds of communication going on between those systems. So if you just log those communication steps, every request from one system to another, that will go a very long way toward letting you narrow down the problem. You can see which step in the chain do we first see errors starting to occur, or if suddenly everything got slow, which of those flows was the one where the performance suddenly shifted?

[0:41:24.4] JM: Do you consider that — Is that a distributed trace? Because I hear about distributed trace being referred to as if I make a call to a service and that service makes a call to another service, you want to be able to trace the error or you trace the different steps that a single call will make through different services, but I think you're talking more generally about different communication points and you want to be able to snapshot whenever there is a communication between two points so that you could collate, just collate that data on the fly. You're not exactly talking about distributed tracing, right?

[0:42:00.8] SN: I think it's two different angles on the same thing. Ultimately, what you want is every time there's a communication from one system to another, you want that to be recorded. Then the question is what you do with the data. Distributed trace is one view of that. Pick a particular request for a browser and let me follow that one back. So it went to the web frontend, then the web frontend communicated with an application server, the application server communicated with the database server. Let me slice through all the different hops that occurred in the process of satisfying that one request from the browser. That can be a very useful thing to look at.

Then taking a completely different angle on the same question as, "Over the last four hours, show me every single time that this application server — I don't care about any of the other application servers, but this one, because this one is acting funny. Show me all the times this application server talked to this database server or talked to any of the database servers," or take some slice like that, and I want to look at statistical, what the average performance of that look like, or the average data transferred or some other metric, or how did the mix of operating — It used to be we're doing 80% gets and 40% puts, but did that shift.

Now, instead of looking at one request at a time and following it all the way through the system, you're focusing on one little box on the architectural diagram or one connection between two boxes and looking at statistical properties that you're looking at the same information, recordings of when one service talks to another service, but you're slicing it on a different dimension.

This goes a little bit back to — I was talking about the 17 different tools we used at Google. A lot of them were just taking — Often, they were ultimately looking at the same data or at least data that have been recorded from the same source, but they were taking very different slices through it, and those are useful for solving different kinds of problems.

[0:43:58.3] JM: The way that your system works, the way that Scalyr works, is people deploy an agent to the different servers that they want to gather log data from. The agent gathers that data and sends it to Scalyr's servers, and there is a managed logging and monitoring system. Give a description for the architecture of Scalyr. Like maybe starting with the agent and the ingress process of that log data once it hits your servers and the indexing. Talk about how that works.

[0:44:34.0] SN: Yes. The agent and truth is quite simple and it somewhat deliberately. We keep as much functionality on our site as possible where we can monitor it and debug it and we don't need to ask customers to do an update if there's a bug there and so forth.

The agent is just a relatively straightforward piece of Python code that just watches log files, notices when they get longer, basically have some smarts for things like understanding log rotation. Basically, just looking for bytes on disk and then it's shoving them up over a HTTPS connection to our server.

Then when we receive the logs, we parse them. You can define rules, “We understand this is JSON or this is a web access log,” but you can build or will build for our customers. Often we’ll do it just for them, whatever funny homegrown application debugging log it is or whatever thing. We write rules that — We say parsing, which is basically extracting the structured data. This chunk of text is the HTTP status code, or this is the error message from the debugging log or this is the customer ID who performed this operation. We pull that out and then we store everything in what's basically a columnar database. So we don't do — Very unusually for a log management, we don't do any indexing, and there’s a couple of reasons for that.

At the end of the day we think this — It's absolutely critical to give people good performance. When you go and you need to ask a question, you just search to your logs. You need to pull some piece of information out of those logs. As we talked about, you want to get it back right away. So it's all about performance, but performance always means price performance, because if you throw enough machines and enough engineers at anything, you can make it faster.

We really think about what's the most cost-effective way to run a system like this and building and maintaining keyword indexes for machine log data turns out to be really expensive and kind of frustrating to a scale — You start trying to deploy a really large elasticsearch cluster for something like this. It certainly can be done, but it gets harder and harder and you have to — And it gets expensive, because if you think about it, the engineering parameters of log data are very different than a lot of the other kinds of — Some are classic document retrieval applications that keyword indexes were designed for. A log message is much smaller than a book or a document. The lifetimes are a lot shorter. So you're returning these huge numbers of small records.

So we don't do any of that indexing. We basically just store the raw data, but we do chop it up into columns according to these parsing rules. So here's all the — From your web access logs, here are all the user agents. Here are all the HTTP status codes and so forth. Then when someone goes to do a search, we basically just do what [inaudible 0:47:32.6] does. We just scan through the data, which sounds dumb — And in some algorithmic theoretical, since it is kind of dumb, but the two things that make it work are, first, because it's a relatively straightforward approach, we've been able to build the whole thing from the ground up ourselves. So there is no off-the-shelf code open source or otherwise in the heart of our storage and search engine. We wrote all of that from scratch, and so it's just very — Often, when you take some general problem and you write your own implementation for your use case where your use case is very specific, you can drastically simplify the problem.

We've built something that's very, very streamlined. You see this coming on other problem domains. I'm not having any examples come to mind, but sometimes you'll read about on Hacker News or something, I'll read about something someone did where they wanted to solve a very specific problem and so they wrote their own database or they wrote their own thing that you should never write your own of. Because their problem was really much simpler than the general problem, sometimes that can make sense, so it's very streamlined.

Then the other thing we —

[0:47:32.6] JM: Falcor. Falcor is an example. Falcor versus GraphQL. Falcor being the most speciric problem statement for Netflix.

[0:48:50.5] SN: I'm not familiar with that one, but —

[0:48:52.4] JM: Okay. Sorry.

[0:48:52.8] SN: Yeah.

[0:48:54.2] JM: It just came to mind as an example.

[0:48:55.2] SN: It came from Netflix, so I suspect they know what they're doing. The other thing we looked at is if you think about how these systems are used, it’s log management, it's very sporadic. It's not like the MySQL database behind some e-commerce site that's hopefully getting traffic 24 hours a day or at least may be 10 or 12 hours a day. You go in usually when there's a problem.

That system is not really being used a lot of the time. By writing a centrally hosted system, we’re able to mix the workload from all over different customers, and so we can basically — When someone goes into our system and does a search, they get all of our hardware. Within a few milliseconds of us receiving that search, literally every CPU core on every server in our cluster is working on that search.

The way sort of the queuing theory of it works out is by the time somebody else, someone else in the world is hitting the button on their search, we’re usually done with the first one. So we get to just work on one search at a time and through the whole cluster edits. We’re just this huge — We’re basically just sort of this huge brute force hammer, and that's how we’re doing the searches.

[0:50:04.3] JM: The columnar database — So I have done some shows about columnar data. I've never worked with it myself, so I'm not super familiar with how columnar databases work, but one understanding I have is that when you when you build a columnar databases, it’s really good at serving things like aggregations, because you could keep — Instead of having your system organized by rose, and if you want to do an aggregation over a row-based database, you have to — The query engine has to skip over all of the places on disk or in memory or whatever where that — Because you have it laid out by row. You're spending a lot of time scanning columns in those rows that don't actually mean anything to you, but the advantage is that you have a lookup table with all of the information for each individual row, and the columnar databases are more useful for doing things like aggregations, like aggregating the total sum, everything that is in a given column. Like if you're doing some of banking transactions, for example.

The columnar database, the ones I've talked about, I think, are not so good at doing individual lookups. So I find it interesting that here you're doing — You're building a system where you're going to need to make individual lookups, because if somebody is looking for a specific piece of data in their logs, they're going to want to search over those logs. Am I stating the problem correctly?

[0:51:36.9] SN: Yeah. I could spend a week talking about all the details of what we do. Actually, we store. We think of it as one extra column that is the full log. So there're sort of two copies of everything. There is a column that has the whole text of each message, and so if you look at that whole column run together, it basically looks just like the original raw log file. So if someone just tells us, "I want to find this error message or this customer ID and I don't even know what field it might be, and maybe we didn't even parse that. I just want to grab. Please grab for me." That's where we'll go.

So then we're not taking advantage of the columnar structure, but we have the data all laid out and we use this, what I was talking about earlier. So we're able to plow through that pretty quickly. The cluster today, we can scan about 1 TB per second when we're just looking through raw logs like that. But then if you tell us, if you give us something a little more specific, like I want to find all the 503 status in the last day and see how that breaks down by URL. Now, we're just scanning the status column and the URL column. So we kind of flip back and forth using the word query optimizer would overly glorify what we've built, but we have a little bit of intelligence where we look at exactly what question you've asked. We'll look at either the raw log or the narrower columns depending on how fits the query.

There're a lot of other things in there. I won't try to take the time to talk about, but the one thing I think is interesting to highlight is there's a whole — I've talked about how these systems are used intermittently. There is a different category of usage that is not intermittent, which is things like a wall-mounted dashboard that you're constantly refreshing or alerting rules, “Tell me if this message shows up in the log. Tell me if 50th percentile latency on this system goes above this level.” You want to check continuously.

The interesting thing about those queries is they’re very repetitive and they’re known in advance. You configure a dashboard or you can configure an alerting rule. Then we know, “Okay. Here's a query we’re going to see over and over and over again 24 hours a day.” We have a whole other system separate from what I've been talking about to handle those queries where we basically create a time series database under the hood just to handle those queries. There are really two sides to the system we’re running.

The high-level idea of, “Let’s look at all the different ways people use their log data, and what's the system we can build in aggregate across all of the things that all of our users are doing 24 hours a day where we can efficiently handle it.” In the cases where, “Oh, the thing we’ve built isn't optimal for that particular use case,” since we've handled a lot of the other traffic, a lot of the other searches very efficiently, then we can fall back on, “Well, if we weren't able to do it in some cheap way, we just let the entire cluster hit it with a hammer, and so then we keep it fast for everyone.”

[0:54:31.4] JM: One way — Just to come back to that, that columnar data store. One way of phrasing the problem statement of building the logging system regarding what you said, is you need to take unstructured data, which is logs — And I guess it's somewhat structured. It has in a certain metadata around the logging message and whatever. But you want to provide some structure to it, but you pointed out that if you just index it naïvely in something like Elasticsearch, it's going to be very expensive to run. So you have to make some trade-offs in terms of how you are organizing that data so that you can look it up while also keeping your costs down a little bit.

I know we’re up against time, but maybe you could just kind of wind us towards a close by contrast the system that you've built with what we’ll call you the most popular logging systems or more popular logging systems, things like Elasticsearch, Logstach Kibana. I think that's the the most popular homegrown logging system that I hear talked about, the Elk Stack. How does Scalyr contrast with the other popular logging solutions?

[0:55:44.0] SN: Yeah. At the end of the day we’re all trying to solve the same problem, and so there’s a certain amount of parallel evolution that happens. In fact, we've occasionally had people at first glance mistake us for Kibana, and which you we didn't particularly Kibana when we’re building this site. There is some parallel evolution going on. But everybody — You have logs you’re ingesting them. They’re being stored. You want to search through them. You want to produce a graph or some other visualization around that or you just want to scroll through the results.

So a lot of it comes down to how much is it going to cost you in dollars for machines and then also in your own effort to set that up and maintain it. People talk about ELK, so you that's three different components; Logstash to process the logs, Elasticsearch to store and search them, Kibana for the user interface. If you want the full package, if you want to have things like alerting rules and depending on where you're pulling your logs in from, getting them out of various Amazon logging services or other places or getting a whole bunch of distributed machines. You have other tools that often are coming in to pull the logs in. You have a lot of these different moving parts that are designed to work together, but they're different and you have to install them separately and you have to configure them to work, configure each one and so forth.

So you just wind up with a lot more moving parts that you have to juggle. Then where it really gets difficult, we found — I should disclaim that this is not something I've ever done myself. We react a lot to what we hear from our customers. Often they're moving off of something like ELK. So I just get anecdotes about people's experiences, but as they’re trying to scale, it starts to get challenging. That you need more and more nodes and then that's more of a management overhead and it's expensive and then you have to start — You get hotspots and you get other kinds of challenges. It just starts to get a lot more complicated to — You can maybe get to the same place, but you have to think a lot more and put a lot more work into keeping it there. Even then, you're going to a have a really hard time getting it to perform at the level you want. I think part of that is just sort of the mismatch between keyword indexes and machine logs. Part of it is trying to run something yourself rather than an economy of scale of satisfying thousands of people at once.

[0:58:05.8] JM: Okay. Last question; describe one difficult engineering challenge that you’ve had to solve while building Scalyr.

[0:58:14.3] SN: I only get one?

[0:58:16.1] JM: You only get one. I mean this has been a really fascinating conversation. I’d love to do another show about — Like dive a little bit deeper, because I had all these questions about the architecture of Scalyr and how your storing logs, how you’re doing logrolling, how you execute a query, blah, blah, blah, blah, blah. So maybe if you’ve got some senior engineer or CTO or even like an SRE or whoever, I deftly want to do another show on this topic. Yeah, maybe just give us some anecdote that is representative of these challenges.

[0:58:44.8] SN: This is ay be a little bit off to the side of what you're thinking about, but when we first launched as a paying service, when we are your getting our first year production customers who are really relying on us, we were still a two-person company. Both engineers, but a two-person engineering team is not a great starting point to be —

[0:59:05.4] JM: Literally, you and your co-found.

[0:59:07.7] SN: Exactly. We had people relying on us 24/7 and we’re both past the age, if there ever is an age, where it's fun getting paged at 3 a.m. One challenge was how do we keep this thing running at the reliability that we want to have without getting woken up all the time? I think we wrote a blog post about this a while back, but we had to put a lot of thought into making sure the system is going to wake us up when there's a problem, but also make sure the system almost never wakes us up.

A lot of that we realized came down to a lot of problems are something that builds up over time. Sort of a dumb example is your disk filled up. It can get much more subtle. Some system gradually approaches its capacity, a queue starts to fill up where the database can't keep up with some query load or something, somewhere saturates. You can see those — A disk doesn't go from empty to full in a moment. A queue doesn't go from empty to full in a moment. So we’ve put a lot of that — It may sound a little bit cute, but we use a lot of Scalyr to monitor Scalyr and keeping out for that kind of thing. So we put a lot of effort into looking for those tipping points and saturation points and making sure we were seeing them a few days before instead of just finding out when it hit capacity at 3 a.m. and things stopped working.

[1:00:35.5] JM: All right well. That's a reasonable place to stop. I look forward to doing more shows about this topic. Steve, you've been an awesome guest. It's been a real pleasure to have you on, talk about this history and the companies that you've built. It's a fascinating discussion.

[1:00:51.3] SN: Thanks. It's been fun for me to. Again, I appreciate obviously putting some work into the questions and that made this really fun and it gives me a chance to share a few war stories, which I always enjoy. Yeah, I'd love to see this happen again.

[1:01:05.5] JM: All right, Steve. Thanks a lot.

[END OF INTERVIEW]

[1:01:09.8] JM: Who do you use for log management? I want to tell you about Scalyr, the first purpose- built log management tool on the market. Most tools on the market utilize text indexing search, and this is great for indexing a book, for example. But if you want to search logs at scale fast, it breaks down. Scalyr built their own database from scratch and the system is fast. Most of the searches take less than a second. In fact 99% of the queries execute in less than a second. That's why companies like OkCupid Giffy and CareerBuilder use Scalyr to build their log management systems.

You can try it today free for 90 days if you go to the promo URL, which is softwareengineeringdaily.com/scalyr, S-C-A-L-Y-R. That softwareengineeringdaily.com/scalyr.

Scalyr was built by one of the founders of Writely, which is the company that became Google docs, and if you know anything about Google Docs' history, it was quite transformational when the product came out. This was a consumer grade UI product that solved many distributed systems problems and had great scalability, which is why it turned into Google Docs. The founder of Writely is now turning his focus to log management, and it has the consumer grade UI. It has the scalability that you would expect from somebody who built Google Docs.

You can use Scalyr to monitor key metrics. You can use it to trigger alerts, it’s got integration with PagerDuty and it's really easy to use. It's really lightning fast, and you can get a free 90-day trial by signing up at softwareengineeringdailycom/S-C-A-L-Y-R, softwareengineeringdaily.com/scalyr, and I really recommend trying out.

I've heard from multiple companies on the show that they use Scalyr and it's been a real differentiator for them. So check out Scalyr, and thanks to Scalyr for being a new sponsor of Software Engineering Daily.

[END]