EPISODE 687

[INTRODUCTION]

[00:00:00] JM: Log messages are fast, high-volume, unstructured data. Logs are often the source of metrics, alerts and dashboards so these critical systems are downstream from a log management system. A log management system needs to be highly available so that a failure in one part of your log management system will not be correlated with the failure of some other part of your system because you're going to be using your logs to triage that system.

Users of a log management system are often building tools based off of the query engine of that log management system. For example, I might build a dashboard that gives me a line graph representing the number of times a certain log message is alerting me due to a memory warning. That way, those memory warnings gets translated into something that is more visual than a big blob of text. I can write a query to return the instances of those memory warnings as they appear in log messages and the line graph that gets displayed is a visual representation of that query. A log management system needs to be able to quickly serve users that are querying their logs, whether it's for dashboards or for ad hoc queries to zoom in on a particular log message that indicate something went wrong.

When logs are in ingested by a log management system, the logs get parsed in a way that can bring some structure to the blob of text that is a raw log message. Some log management systems will then add the log message to an index. An index can allow for very fast lookups of particular types of queries, but an index also has certain constraints, such as processing regular expression queries.

Steve Newman is the CEO and founder of Scalyr, a log management system that uses a column-oriented data storage system instead of the more conventional index-based log management systems. Today's episode is a great case study in distributed systems tradeoffs. Steve talks in great detail about how Scalyr maintains high uptime and its system for ingesting logs and serving queries.

Before we get started with this episode, I want to mention that we are hiring for a couple of roles including writers and podcaster. If you are interested in those roles or operational roles, we have a number of job listings at softwareengineeringdaily.com/jobs. I'd love to see your application.

[SPONSOR MESSAGE]

[00:02:38] JM: Kubernetes can be difficult. Container networking, storage, disaster recovery, these are issues that you would rather not have to figure out alone. Mesosphere's Kubernetes-as-a-service provides single click Kubernetes deployment with simple management, security features and high availability to make your Kubernetes deployments easy. You can find out more about Mesosphere's Kubernetes-as-a-service by going to softwareengineeringdaily.com/mesosphere.

Mesosphere's Kubernetes-as-a-service heals itself when it detects a problem with the state of the cluster. So you don't have to worry about your cluster going down, and they make it easy to install monitoring and logging and other tooling alongside your Kubernetes cluster. With one click install, there's additional tooling like Prometheus, Linkerd, Jenkins and any of the services in the service catalog. Mesosphere is built to make multi-cloud, hybrid-cloud and edge computing easier.

To find out how Mesosphere's Kubernetes-as-a-service can help you easily deploy Kubernetes, you can check out softwareengineeringdaily.com/mesosphere, and it would support Software Engineering Daily as well.

One reason I am a big fan of Mesosphere is that one of the founders, Ben Hindman, is one of the first people I interviewed about software engineering back when I was a host on Software Engineering Radio, and he was so good and so generous with his explanations of various distributed systems concepts, and this was back four or five years ago when some of the applied distributed systems material was a little more scant in the marketplace. It was harder to find information about distributed systems in production, and he was one of the people that was evangelizing it and talking about it and obviously building it in Apache Mesos. So I'm really happy to have Mesosphere as a sponsor, and if you want to check out Mesosphere and support Software Engineering Daily, go to softwareengineeringdaily.com/mesosphere.

[INTERVIEW]

[00:04:57] JM: Steve Newman, welcome back to Software Engineering Daily.

[00:05:00] SN: Great to be here. A lot of fun last time.

[00:05:02] JM: Yes, although last time we mostly talked about the engineering of the first version of Google Docs and your experiences at Google, and there're some really great stories there, but we didn't get to talking about Scalyr as much as I would have liked. Scalyr is the company that you're building around log management.

The topic I want to start with is log management. Log management is the process of taking in logs, indexing them, building metrics around them. This has been a problem that has been open to engineers to solve for a very long time. Why does it continue to be a thriving industry? Why is log management an unsolved problem?

[00:05:45] SN: The short version is because it's a hard problem and always kind of new frontiers to push forward on, and at a high-level there's two things that make it hard. One is just scale. Often, there're massive amounts of data to work with, and ideally you want to be able to search through that data interactively. So you just got large scale data and you're trying to get answers out of it quickly.

The other problem is I might summarize as heterogeneity. We talk about logs sort of as one concept, but there are lots of different kinds of logs, and not just different formats, but very different uses. A web access log is very structured thing. It is a very regular kind of data that's very standardized across a lot of different environments. An application debugging log is a jumble of very specific messages that have been sort of thrown together ad hoc. There's nothing organized or coherent about it. It's unique to the particular application and to the engineer or actually succession of engineers who have had a role in creating that, and it's different for every application.

It's really lots of different problems. You do some structured analysis on my very structured web access logs, do some kind of ad hoc exploration through my debugging information. In a single problem session, you may be working with a lot of different logs from a lot of different systems. Every customer or every system has a different situation with a different mix of systems and logs and configurations. Log management is a whole subfield of its own. There're some fundamental difficulties around scale and performance, but then also this huge heterogeneity means there's not just one problem to focus on.

[00:07:30] JM: Yeah. The schema of different logs in different contexts can widely vary, and then there's also heterogeneity in terms of how important different logs are. You've got all these knobs that you could potentially present to a user and say, "Hey, do you want to tune this knob? Do you want faster access time? Do you want better indexing?" But the user may not even want to configure their log management system. They just want something that works out of the box. Then you have all these questions also in the context of really high velocity, like logs just don't ever stop coming. There's constant data.

[00:08:09] SN: Yeah.

[00:08:10] JM: So if I'm an engineer building a new startup, I think we've pretty much laid this out, but maybe make the case even stronger, why is log management something that I want to buy as a service rather than rolling my own? Because there are people that still roll their own log management systems.

[00:08:31] SN: Yeah, and I think there's two reasons. One is just the classic why would you ever not roll your own of any – It's just a classic SaaS argument. Focus on your core competency. Let a log management expert worry about your log management and you worry about your business. Don't devote engineering resources to building and maintaining it. Don't have your own thing that you have to train everyone how to use, because it's not standard and no one's ever seen it before. I don't think there's necessarily anything – I think those are arguments apply here and are not really in a different way for a log management system. Just kind of the classic SaaS argument.

The other thing I think though that's interesting is around scale and performance where it's expensive to provision enough hardware to manage your own logs at high-performance. Whereas if you work with a hosted service, at least if it's architected appropriately, there's sort of an economy of scale there because your use of these services tends to be bursty. You're troubleshooting a problem and then you're not. Then the next issue comes up or the next thing. So you're going in and out of your log management solution. Even with a large team, with hundreds or thousands of engineers, the usage is still very bursty and intermittent. So there's a real efficiency from pulling together a lot of different uses in a central service rather than having lots of little islands of individual log management.

[00:09:53] JM: So if I understand you correctly, when you piggyback on a log management system, presumably that log management system is hosting a lot of people's logs and the access patterns for logs are such that you have a particular type of – Maybe you want to call it caching, or availability of these logs where if I'm solving a problem right now, I want my recent logs or the logs that are related to a certain topic, a certain problem I'm trying to solve, I want those logs higher in the cache hierarchy. I want to be able to access them faster. But if I'm not solving that problem right now, I can have those logs on disk and in a place that is not a faster access time and their economies of scale to that type of infrastructure is what I'm hearing from you.

[00:10:47] SN: That's right. Also, and maybe even more importantly, is around the – So the storage and RAM and higher levels of the cache hierarchy are important, but processing is also very important. Even if you have data an RAM, when you want to go search it or query it, that takes up – There's a big burst of processing activity that happens there. For half a second, you're pounding the heck out of a bunch of CPUs and then the half second is over your search is done, and for the next 7 seconds, maybe you don't need the system at all because you're thinking about where to go next. So that processing capacity is also something that can – There's a lot of efficiency in sharing that where you have a collection of processing capacity that's constantly turning its attention to whoever needs literally in that millisecond.

[00:11:33] JM: What is the process – We talked about this a little bit in the last episode, but this is like a deeply complex topic that your infrastructure is all about consuming and indexing and storing logs. Give me an overview of the path of a log message as it gets ingested by Scalyr.

[00:11:51] SN: So it depends a little bit depending on where that's coming from, but let's take the most common scenario, which is one of our customers is running some application, that application is writing a log file to disk on the customer's server container or whatever, and our agent is sitting there waiting to pick that up.

So the application appends the message to a disk file. Our agent is sitting there tailing that file. So it notices there are new bytes on the end of the file. It picks those up, throws them into a local buffer and typically about once per second it will package that buffer up and send it off to our backend. From the agent's point of view, it's just throwing that block of data at www.scalyr.com.

On our end, that goes through a proxy layer and lands at a queue server. This is a pretty simple system that we've built that just takes in these little bundles of data that are constantly flowing in from all over the globe, from all over various customer systems. The purpose of that tier is really just to smooth out any little hiccups in the main processing stage. The data rise on our end and we can immediately throw it in the queue, acknowledge it, and the agent knows that its work is done, that log has been successfully received at the Scalyr end. It sits in the queuing layer typically again just for another one or two seconds, and then it goes to what we call a backend node, which is the kind of the heart of our system. This is where the logs are going to be parsed, stored and held for querying. So everything from the ingestion through the querying happens in that backend node.

The bundle – That bundle is whatever new collection of your log message have arrived at the end of that file in the last second or two, it might be one message, it might be 100,000 messages, but that bundle arrives at the backend, which parses it and it's appended into – I think last time we talked a little bit about how we basically have a columnar data storage system that we've build. If it's a web access log, you've got the IP address, the URL, the status code, all these different fields of the logs become columns on disk. All of those messages get appended to the relevant columns. Now it's on disk and cached in memory on that backend and it's available for the next query.

[00:14:01] JM: I want to walk through some of those stages. So the ingest of the logs starts by hitting a proxy layer. The proxy layer forwards it to this queuing layer, and you said you wrote your own queuing layer. Is it on top of some kind of queuing system, like a Redis queue, or a Kafka queue, or something, or completely written from scratch?

[00:14:24] SN: Completely written from scratch, and the truth is it was a fairly small project. The reason it was a small project is a very, very specific, fairly straightforward requirements, but the other half of it is kind of the reason we did chose to build our own implementation is to make sure that we could meet all of those requirements that we had and do it in a very efficient way.

One of the reasons that it was relatively straightforward is we have a lower level storage engine that we had already built for the backends. It's based on Google's LevelDB project. So it's a fairly simple storage manager, but it's happened to be a very good fit for a queuing system, basically just throw blocks of data into the storage engine and then you can – One of the things about a LevelDB style system, a log structured merge style system, is it's very efficient at random writes and then it scans. Those are the two kinds of data access for a queuing system. Some of the properties that we wanted the queuing tier to satisfy other than simple efficiency was it's important that it maintain in order processing within – So we have lots and lots of streams of data. Each stream basically represents a log file and it's very important that data within any stream be delivered in order, but you may have an enormous number of streams and they need to not block one another. So you need to be able to process different streams in parallel, and we had specific requirements over kind of where data comes from and where data goes to. Which backend nodes, any particular log file winds up on very, very short queuing times and so forth. It was just easy to build a simple little layer from scratch that would meet all of those requirements.

[00:16:06] JM: Do you have redundancy at that layer?

[00:16:08] SN: We do not have storage redundancy. So data is vulnerable while it's sitting in that layer, but typically it's only there for about one second. But if a node fails, then the system routes around that, and that one second worth of data is trapped or a few seconds worth of data is trapped until we recover that node. Meanwhile, everything will just start to route around it.

[00:16:27] JM: Is the recovery, does that need the agent on the client side to be aware and to do some act and say, "Did you receive my last package of data? Was it fully received?" How do you recover from the process of the agent sending to the queuing system and then a failure occurring at the queuing layer?

[00:16:45] SN: We have to put back out of the queuing layer. If a queue node is lost irretrievably, then that one or two seconds of data that was in that – Had already been acknowledge by that queuing layer, but not yet delivered on, that data would be at risk. We've thought about building redundancy at that layer, but the truth is that a hard failure like that is so rare, and when you multiply – You start to look at the kind of number of nines of data retention that you still have there. It just hasn't been a concern for us.

[00:17:18] JM: Are those just built on regular EC2 nodes or using –

[00:17:22] SN: Yeah.

[00:17:23] JM: Okay. Okay.

[00:17:24] SN: Yeah, and our experience is at the meantime between hard failure of an EC2 node is measured in a significant number of years. So it turns into 8 or 9 nines reliability if you divide that into the one or two second window where data is captive in that node.

[00:17:40] JM: Because it's because you're actually writing those logs to disk on that node?

[00:17:45] SN: Yup. Yup. SSD.

[00:17:47] JM: Right. Right. Okay. So you write them to disk, and then even if the machine restarts, it's not problematic. It would only be problematic if the disk completely failed.

[00:17:55] SN: Exactly. Yeah. It's just an example of trying to be very thoughtful about the tradeoffs. Reliability is important, but 9 nines is probably sufficient and sometimes something that sort of theoretically, "Wait, you're only storing one copy of that data, but well, we're only storing it – It's only sleeping there for about one second."

So if you take a careful look at sort of the engineering tradeoffs, then sometimes you can get away with things in a particular situation that in other context might not be as good an idea.

[00:18:25] JM: Then I was looking at your architecture diagrams. I think that the next phase in the log processing pipeline is that it goes from the queue and it gets written to two separate backing stores. Is that right?

[00:18:39] SN: That's right. Technically, three, but often we only talk about two. So it goes to three backing stores. Two of which are sort of high-powered nodes with SSDs. Currently, we're using I3.4 extra-large EC2 instances. Those do all the work. They're responsible for queries and so forth. Two copies is enough for kind of processing in normal healthy mode operation, but to make sure that the data is very durable, we wanted to have a third copy on EBS. So that's what that third copy is. We often gloss over the third copy because the EBS performance and cost performance is not good enough for us to use that in query. So that third copy is just sort of for emergency purposes and isn't interesting to most of the system, but it does exist.

[00:19:25] JM: How much cheaper is EBS than SSD?

[00:19:27] SN: If you look at the price per gigabyte of EBS versus take something like an I3.4 extra-large node, and when you consider the full cost of that node, it's a lot higher. Now, you're also getting a lot of processor on RAM and so forth, but basically we would arrange our SSD nodes in pairs rather than in triples because doing them in pairs gives us a better ratio of processing in RAM to storage.

[00:19:58] JM: I guess I'm having trouble understanding. So you have two – So the architecture is, for the main replica set, the replicas of your – You want three copies of the log data, like the source of truth data, the customer data, this is your business. So you need three of them, and you have two of them on SSDs, the two main copies, and you're periodically writing to the third copy so that you have a failover in extreme cases. I didn't quite understand what you're saying about you like to keep the SSDs in pairs because of processing something?

[00:20:30] SN: Basically, every byte of SSD storage comes with a lot of RAM and processes attached. So storing three copies on SSD means we need three shares – It means we're also paying for that much more RAM and CPU and it's more than we need.

[00:20:50] JM: Totally. Totally. It makes complete sense. Like just as a cost savings mechanism. So I guess we've gotten from the queuing place to the point of writing to the actual log storage system. What happens in that process of a write from the queuing system to the storage system?

[00:21:10] SN: First of all, that's where we do parsing, and by parsing we mean that stage where we look at this chunk of text and start figuring out, "Okay, these 11 bytes are the HTTP header, and these 39 bytes are the URL, and these other bytes are the refer," and so forth. So this is where we're identifying the structured content of the log. So that's a whole system that we can talk about.

So we do that, and so now we think of it as just a collection of key value pairs, "This message had the following text and it had a field named status, which was 502 and it had a field named refer, which was whatever," and so forth. We take those collection of values and we – This is all happening locally within one EC2 instance, which is that backend node. We've parsed it, we've got this collection of fields and we append it on to our columnar data store [inaudible 00:22:05] slightly lower level means we have a whole collection of values, one value for each field that are getting to written to N different places in the key value store, the LevelDB like system.

So all of the log messages that were in that batch, all of the fields and the various messages in that batch, that whole collection of stuff gets written out as one database append transaction and winds up on the SSD and also in a memory layer, and then it's immediately available for the next query that runs will see it.

[SPONSOR MESSAGE]

[00:22:47] JM: I have learned a ton from QCon. QCon San Francisco takes place this year, November 5th through 9th, 2018, and I will be going for my fourth year in a row. I always love going and seeing the talks, and in between the talks I hang out and eat some mixed nuts, chat

with other engineering leadership about the latest talks and stuff that they're seeing, the 50 different stream processing systems that we're seeing, the different databases we're seeing, and QCon is a place where senior software developers and team leaders and managers and senior leaders, they all gather together, and you have fantastic conversations. You have fantastic presentations, and it's extremely high quality. Its technical. A lot of it is also cultural and about management, and you can get $100 by using the code SED100.

QCon is a great learning experience. The presentations this year include 18 editorial tracks with more than 140 speakers from places like Uber, Google, Dropbox, Slack, Twitter. They are curated high quality talks, and you can get $100 off if you use code SED100 at checkout for your ticket. Again, QCon San Francisco takes place November 5th through 9th, 2018, and the conference is the 4th through 7th, November 8th through 9th are the workshops. You can go to qconsf.com to find out more.

Thank you to QCon. If you are going to buy a ticket, please use code SED100, and we really appreciate the sponsorship of QCon.

[INTERVIEW CONTINUED]

[00:24:45] JM: The parsing part is that first step that you mentioned. Parsing, does that require an understanding of the schema of a particular type of log message, or is the parsing phase agnostic of what company and what source the log is coming from?

[00:25:04] SN: It does require some knowledge. This is a problem where you've been working on for the whole 6 or 7 years we've been in operation, and we've gone through a few iterations on how we approach it. The highest level is kind of two forks in the road. You can basically take an ML approach or a manual approach, and we've actually found that what works best is to keep a person in the loop, define the parsing roles manually and just build a system that makes it really, really easy to do, because if you look at a typical log file in practice, it's hard to define a machine rule that can take any random log file and make a sensible idea of what to do with it.

But when a human being looks at it, more of often than not, they're actually very, very simple. You've got 9 fields and there they are, boom, boom, boom, boom with spaces in between or

sometimes it's commas, or it's a printf message. So you got three words that are part of the printf template, and then you've got your first field, and then you've got six more words that were from the printf and another. So it's very, very simple. There's some stuff that isn't the field, and then there's a field, and then there's some more stuff that isn't the field, error on line, line number is a field, error messages of field, unwrapping stack. Unwrapping stack is not a field, but the error message right before it is a field. So it's very easy for a person to look at it and say, "Okay, these three pieces are fields and the rest of it is just always going to be the same."

So we've built a system where there's a little interactive editor where you can define the rules, and the rules kind of classically, when people build systems, they allow you to define these [inaudible 00:26:46] rules by hand. They tend to rely on regular expressions. Like you have to create a regular expression for each field of the log, and the data goes in those fields, whether it's a URL, or an email address, or an IP address, or whatever. It tends to be a little bit complicated. But the stuff in between tends to be very simple. Between each field, you have a space, or you have these three words from the log message then it's always those same three words. It tends to be very, very simple.

So we've built a system that's sort of inside out, where instead of giving regular expressions for the fields, you give regular expressions for what's in between the fields. Usually those are just trivial. Again, it's just a space character or something. So we've just gone the route of making that really easy to set up. Then to sort of take advantage of that, we actually put a button in the product that says, "Build it for me." That doesn't invoke an AI, it invokes our support team. It just sends a message to the support team and we sort of streamline the workflow. So when you click that button, our support team gets a little sample of your log. They get any note that there's a place for the customer to attach a note if there's anything they want to say about it and on our side it drops us right into a little workflow. With that interactive took, our customers also have access to the tool, but often they don't bother. They just click the button and we take care of it. So it's not much work for us, and it's frankly a little bit of a hack where we get to look like we've done somebody a big favor, but it's actually not that much work.

[00:28:12] JM: Yeah. Because how many different logs schemas as a given company going to have? Well, I guess they could potentially have a lot, but typically you have an engineer and

they're working on a service and they're working on that service for like three months and that requires one point of setup for their particular log messages. So I guess it's not a huge –

[00:28:30] SN: Exactly. Yeah, that's really kind of the flavor. Also, a lot of it is just standard, "Oh, that's a web access log." "Oh, that's a PostgreS query login," and we've just got all those on the shelf.

[00:28:39] JM: Okay. Cool. So you can get that schema done from a combination of basically human labeling and some aspects of logs that are easier to identify, and an understanding of that schema lets you map out how to ingest, how to parse a log message. Then you parse it into some kind of intermediate data structure and then you take that intermediate data structure and – What? Do indexing and stuff on it, or what's next?

[00:29:11] SN: So this is maybe the thing that people find most surprising about our approaches is we don't do indexing, actually. I'm glossing over a few details, but basically the core of the system, there is no indexing. Once we've done that parsing and we laid the data out in columns, that's sort of the most interesting part about what we're doing with the data ingestion time.

So we get a batch of new messages on the end of your web access log, we parse it out, and now consider the refer column. So we've got the refers for each of these. Let's say it was 80 new log messages. So on disk in the area where we're storing the refer column of this particular log, we're just going to write those 80 URLs out one after the other, boom, boom, just as almost like a dumb text file, and that's it. No indexing.

So then if a customer says, "Find me all my traffic that was referred from Wikipedia," we're basically just going to grep that column, that region of the disk where we've stored all the URLs. Not literally grep, but our own internal code that's ultimately running the same inter-loop pretty much that a grep implementation might run.

So we don't have an index that can tell us, "Oh! Out of these 3 million log messages, here are the 9 that have the word Wikipedia." We have to scan through all million refers, but it turns out that if you're streamlined about how you do that and you're not kind of waiting through 9 layers

of legacy implementations built on top of one another, it's actually just doesn't take that long to stand through a bunch of data.

One of the key numbers, metrics that we think about is – I can't remember whether we got into this at all last time, but one CPU core on our backend system can scan about a gigabyte of data per second. Especially if you're just scanning one column, like the refers as supposed to the entire log, a gigabyte is a lot. Then you multiply it by thousands of cores, that's the heart of where our performance comes from.

[00:31:08] JM: So you could parallelize the scan.

[00:31:09] SN: Exactly. Yeah, the other number that we think about is the aggregate performance of the whole cluster. So right now it's about 1.5 terabytes per second. Meaning, we have about 1,500 cores in this cluster. Each one can scan about a gigabyte of data per second. So if a query involves scanning half a terabyte of data, then we can do that in about a third of a second.

[00:31:30] JM: Your columnar storage system, what are in the columns? A given record is a column, and then the column has all of the text of one of the fields in the define schema. Is that right?

[00:31:45] SN: Exactly. Again, take the refer field. There'll be a region on the disk that's just URL after URL after URL holding the refers and the successive log messages, or the status column. There'll be a comment that just says, 404, 502, 200, 200, 200, 503.

[00:32:04] JM: I guess that makes sense, because if you think about it from the access pattern of an engineer that's looking through logs, they know the field that they want to be looking at. They know that I want to find logs where the refer was www.walmare.com.

[00:32:21] SN: Exactly. Yeah. Very often, it's like that. Find that, or find me on my 502, or on my 404s, or everything where the user agent was Google bot or like that.

[00:32:33] JM: Are there use cases where they would want to just like, "I just want to type in Wikipedia and I want to scan every column of every log message."

[00:32:44] SN: Yeah, that actually happens all the time, because people are, I'll say efficient, and I mean it. Why should you bother instructing the clever query when you don't – If the tool can accommodate you?

[00:32:57] JM: Yeah, you got enough cores. Why not?

[00:32:58] SN: So then we just have to scan the whole log, but that's still pretty fast. One of the properties of the system we've built, an index-based system can always be more efficient for the types of queries that the index was optimized for. But the flipside of that is it can become woefully inefficient for queries that it was not optimized for. If you're relying on that trick of, "That's got the keywords index, and if you want to look for a keyword, then I'm all set." Then as soon as you're doing something that isn't looking for a keyword, like you're looking for a regular expression or you're doing a numeric comparison. Show me everything where the time was more than 100 millisecond. So I need to find 100 or 101 or 102 or 103, and there's an infinite number of values that should match. So using the keyword index, someone becomes a lot harder.

So the system we've built, because it's very simple, it basically just scan the data and the variations mostly build on do we have to – Sometimes we have to scan all the data and sometimes we only have to scan certain columns. The performance is going to be pretty consistent.

So there's some queries where an index will you back an answer in one microsecond, because you just look up the index, there's only one match. There it is. Boom! Done. We will never complete a query in one microsecond, but we complete 96% of our queries in one second or less. Rather than trying to just be like absolutely fast on some stuff, and then a long tail, we just sort of aim down the middle for everything.

[00:34:27] JM: That's a great compromise, because if you think about the regx case, you can parallelize scanning whatever columns you want with a regx and you contrast that with trying to

have a regx hit against an index based system. How does that even work? That sounds like it would be slow or it would be hard or the system is not built for it, right?

[00:34:51] SN: Exactly. Depending on how clever you are, sometimes you can notice, “All right, in the middle of this regx, there’re these nine characters and it can’t match any string unless the string has those nine characters. So I can narrow it down a little bit using the –” But ultimately, you’re going to have to fetch all the records and run the regx on them, and that’s what we do, is fetch all the records and run the regx on them. But we’re highly, highly optimized for that, and in a keyword index based system, that’s typically not an optimized path.

[00:35:20] JM: That’s great. Some really good compromises there. In terms of – Okay, we’ve talked about the parsing. It’s not indexing. You just write these columnar messages to columnar storage. What kinds of failures can occur in that parsing and columnar preparation and writing process? Are there any frequent failures or frequent challenges in that process?

[00:35:44] SN: The truth is it’s generally very reliable. I mean, you can always have a node failure, hardware failure or something at the EC2 level, and one of those nodes vanishes. That’s probably actually the most common failure mode we have and that’s where the pairing up of the backend nodes comes in and we just flip over to the other member of the pair.

Maybe over time, another category of issue we’ve had, but knock on wood, I think we’ve done a pretty good job of kind of stamping these out overtime, when you’re running a large scale system like this with lots of customers in you’re very heterogeneous workloads, you run into all the edge cases that you never thought would come up. So you’ve got one log message that’s 50 kilobytes long, or you’ve got a thousand different fields in one log message, or things like that, or you’ve got a single log that’s getting a megabyte per second appended to it. Overall, there’s plenty of capacity, but this one log is trying to squeeze through, because it’s a single log and it’s trying to squeeze through a single backend node. That one node, one thread on that one node can’t keep up with the parsing.

So we’ve kind of overtime build in the safety limits and the strategies for splitting large things up, whether it’s a large number of fields, or large messages or whatever. So kind of early in our

history, those were sort of the interesting things that would happen. Nowadays, a system is mostly just kind of runs.

[00:37:11] JM: That's awesome. That sounds like a good product to be in charge of.

[00:37:16] SN: Yeah. It's been fun – And we get to practice what we preach a lot. We monitor the heck out of our own system. The business we're in, it would be very embarrassing if we were having a problem and we didn't understand why. We monitor the heck out of the system and then we take advantage of that to – When a problem like that would come up, kind of fix the system so that that can never happen again. We also try to put in a lot of just like early warning sign monitoring. If there's some buffer where we assumed early on maybe that there were – This isn't quite a literal example, but let's say we had assumed early on that no log message would ever have more than a thousand fields. Well, then we'd put in some logging, how many fields are there, and we'd put in alerting rules of if ever a customer has a log with more than 500 fields, let us know so we can go in and think about what to do about it before they've hit the limit, rather than after they've hit the limit and it's an active problem.

[00:38:11] JM: That facet of your product where you use Scalyr to monitor Scalyr. So you have another instance of Scalyr on staging. You've got a staging instance, and that monitors your production cluster, and then you use your production cluster to monitor staging. Is that a circular dependency at all? Are there any problems with having that setup?

[00:38:33] SN: So, generally, no. The biggest problem frankly is it's confusing. For example, something that will happen is – Now, the staging cluster has significantly more incidence of problems than the production cluster, because we're deliberately using it as the guinea pig for a new code and we don't provision it as heavily and we don't take quite as good care of it.

So it's a not known scenario, is that the paging cluster will start screaming bloody murder and – Sorry, the staging cluster. That's a Freudian slip, because the staging cluster will start setting up our pagers frantically claiming that half the production fleet has gone down, and clearly there's some kind of horrible crisis going on in EC2 and we've got to respond immediately.

What's actually happening is the production cluster is absolutely fine, but there's some glitch in maybe the queuing layer of the staging cluster, and the reason it's claiming that all those production nodes are down is that it is failing to process the incoming data from those production nodes.

So whenever either system sends us an alert complaining about the other system, we have to decide, "Okay, is that real or is that the crying wolf?" Because, actually, the staging cluster is misreading the situation. Even once you know in principle that that's something you need to think about it, it gets a little bit hard to wrap your head around. Also, every time I want to learn about the production cluster, I need to log in to staging to run the query. Every time I want to learn about staging, I have to log into the production cluster. Sort of that confusion is the hardest problem we've had.

I don't think we've ever in those six years has simultaneous failures of both systems, and there's no particular reason they should be, because we push code on different schedules, and just in general they're entirely disconnected systems.

[SPONSOR MESSAGE]

[00:40:27] JM: Cloud computing can get expensive. If you're spending too much money on your cloud infrastructure, check out DoIT International. DoIT International helps startups optimize the cost of their workloads across Google Cloud and AWS so that the startups can spend more time building their new software and less time reducing their cost.

DoIT international helps clients optimize their costs, and if your cloud bill is over $10,000 per month, you can get a free cost optimization assessment by going to D-O-I-T-I-N-T-L.com/sedaily. That's a D-O-I-T-I-N-T-L.com/sedaily. This assessment will show you how you can save money on your cloud, and DoIT International is offering it to our listeners for free. They normally charge $5,000 for this assessment, but DoIT International is offering it free to listeners of the show with more than $10,000 in monthly spend. If you don't know whether or not you're spending $10,000, if your company is that big, there's a good chance you're spending $10,000. So maybe go ask somebody else in the finance department.

DoIT International is a company that's made up of experts in cloud engineering and optimization. They can help you run your infrastructure more efficiently by helping you use commitments, spot instances, rightsizing and unique purchasing techniques. This to me sounds extremely domain specific. So it makes sense to me from that perspective to hire a team of people who can help you figure out how to implement these techniques.

DoIT International can help you write more efficient code. They can help you build more efficient infrastructure. They also have their own custom software that they've written, which is a complete cost optimization platform for Google cloud, and that's available at reoptimize.io as a free service if you want check out what DoIT International is capable of building.

DoIT International art experts in cloud cost optimization, and if you're spending more than $10,000, you can get a free assessment by going to D-O-I-T-I-N-T-L.com/sedaily and see how much money you can save on your cloud deployment.

[INTERVIEW]

[00:42:52] JM: You talked about the bursty workloads earlier, or I guess bursty in terms of customer use cases. If I'm a customer not regularly going – Or hopefully I'm not working at a company where I have to regularly – here's my regularly scheduled checking of this particular query. I mean, I guess you would want to do that in terms of dashboarding where you always have – That's just a constant workload. If you've built the dashboard within Scalyr and you've got a query that's basically constantly running against it. I don't know, we could get an engineering of dashboards. How bursty is the consumption of usage across different users on your infrastructure? Is it bursty product wide or just like by customer?

[00:43:39] SN: Mostly by customer. Certainly, within an individual costumer can be very bursty. A classic scenario would be if they're having some big problem, whether it's an infrastructure problem or they push the code release or whatever, or – And this is a real example. We've seen it happen. Some of our customers are e-commerce sites and on Black Friday their traffic may octuple. First of all it just means they're sending us 8 times as much log data, but also that's probably stressing their system out and they're all hands on deck at their end and suddenly all their engineers are logging to Scalyr and running in different queries all intermixed.

So at the individual customer level, it's very bursty. But we have a wide variety or customers. They're not all e-commerce sites. They're not all located in the same geographic region, so they have different business hours. Also, by the same token, say there's an AWS infrastructure issue that may affect a bunch of our customers, but not all of them, because they're not all on the same regions or not all of their infrastructure is in the same region. The overall set of customers that we have, they're all over the world, they're on all different businesses and so forth and it tends to even out pretty well. For example, on Black Friday, some of our customers are octupling. Our overall traffic might go up by 20% or 30%

[00:44:54] JM: How does capacity planning or strategic buying of infrastructure? Does that fit in to your work at all?

[00:45:03] SN: It does. The infrastructure isn't cheap, and so that's something we need to monitor closely. This is where that heterogeneity across customers helps us a lot. The processing load tends to even out pretty well, and storage load, just how much data are receiving and having to store. Sort of by definition, that can't change on a dime. You can't on a one day period suddenly have to twice as much accumulated data. So we're able to just sort of watch the trends and be reactive, "Oh, it looks like we've closed some deals, or some of our customers are growing, or whatever. Let's add another 10 pairs of backend nodes." It hasn't been a difficult problem. We just sort of do this straightforward thing.

[00:45:43] JM: Straightforward thing. There are some vendors of products that help do cost management and how to buy infrastructure at the right time. Is that at all something that's like hard to do? Does it get into the world of like automated trading where you have to be snapping up the right instances that are available, stuff like that, or it's just not a concern for you?

[00:46:06] SN: For us, it's not much of a concern. I think that boils down to the fact that at an architectural bot diagram level, our system is actually pretty simple. We only have about three or four kinds of nodes and the workload doesn't swing around too drastically because it's this mix of all these different workloads. My intuition would be that those products become interesting when the system is too complicated for one person to keep track of, and also the organization is too complicated for there to be any one person who has a good picture of everything. So that's

when you need a machine keeping an eye out for you. But we're just not at that level of complexity yet.

[00:46:41] JM: I want to talk about the business a little bit. I had this conversation with an investor a while ago and it just stuck with me because it contained a simple truth that I just hadn't really considered, but he was talking about investing in – I think it was a monitoring business and I was like, "Why would you want to invest in a monitoring business? It's not winner-take-all. You just have a dog fight for customers and there's just tons of monitoring. There's not really network effects, except the customers kind of talk to each other, or maybe you have like an integrations thing and network effect." But in actuality, there's a lot of winners. So it's not necessarily a problematic investment, because you just have to get some requisite number of customers and you have good economics for every customer. Log management is kind of like that.

I'm fascinated by these business, like monitoring business and log management business, because it seems like every year there's a new suite of them and they all do fine as long as they have a decent product and they get some customers. So how do you think about that widely variable multi-competitor landscape, or do you think of yourself as having some clear differentiator, or does it just come down to marketing? Now that you've got the product so well-tuned and it works really well, what's your process of differentiating?

[00:48:00] SN: Yeah. So it's a great question that we get asked a lot, like, "Isn't this a really crowded market? Why did you want to come here?" I guess the answer connects to the question of differentiation. It really doesn't feel crowded to us, because our key differentiation has always been the performance and usability of the system. We've been talking a lot about how the system is built, and ultimately the main reason we took this whole approach, we're not using in-keyword indexes and so forth. That was a lot more work for us to build this thing. We couldn't just slap down on an Elasticsearch layer and start building on top of that. We had to do all that extra work.

But what it gave us is this interactive performance that 96% of queries run in less than one second. What that means to the user is that it feels interactive. It feels – There's sort of a lightness to the tool where it's like Google search. You just sort of go in and poke at it. You try

this. You try that. You don't have to get emotionally invested in each query, "All right. Let me think carefully about exactly what question I want to ask, because it's going to cost me five minutes to ask it." You just sort of go in there and flail.

That draws people into the product a lot more. It makes it feel easier to use and easier to learn because you're not really punished for your early mistakes where you haven't quite learned the query language or you don't just haven't quite learn how to think about what you want to actually find in the logs.

That's been our key differentiation, is the performance and the ease of use. We've done a lot of work around the UI design to make this easier to use. It's sort of boring to talk about. It's a lot more interesting to show. But I don't always talk about it so much, but I talk about all the iteration we did on our parsing engine, because we've dealt with so many different logs over the years. We've also done a lot of iteration on the user interface so that people don't really have to learn a query language to use the product. You can just sort of click on things and drill down and the query gets built up for you. That also relies on the speed of the query engine. That UI is backed by the fact that we can run a lot of sort of speculative queries, "Oh, let me just do a bunch of analysis around the set – On the fly around this set of logs the customer is looking at and give them bunch of things to click on and drill down."

So to get to your question, the differentiation that gives us is that just a lot faster and easier to use, and what we find is people actually do use it a lot more. Sometimes log management tools will just sort of gather dust on a shelf because it's slow and hard to learn. So that's our differentiation and it's pretty consistent across all the competition. We don't have to say, "Well, on these products, we compete on speed." Again, those products we compete because we have this feature. It's just been kind of simple for us. It's all about speed and ease of use against the field.

[00:50:47] JM: I had a show fairly recently with Fangjin from Imply, which is built on the Druid database. Have you looked at that system at all?

[00:50:58] SN: No. I mean, I've heard of it, but I can't claim to be an expert.

[00:51:01] JM: Yeah. It was interesting talking to him about differentiating in this field. What I wonder is what you – Because we didn't even talk about dashboarding very much, but when you think about applications that you could potentially build on that data storage – That sounds like really the mote that you've built is that ability to scan, just the columnar records, the ability to scan, the ability to parallelize that scan. I can't imagine that code was easy to write. That's probably some stuff that was hard to write. Are there other applications that could be built on that kind of storage layer?

[00:51:34] SN: Probably so, and we really have not explored that and don't intend to explore it on our planning horizon because there's a lot more to a product than just that. But I think fundamentally, where this engine we've built is applicable. One of the important parts about it, we've talked less about this, but the fact that it is multitenant, the fact that we're able to share all that horsepower across a lot of different customers. There's a whole management layer. We've had to build there to make sure – Basically to make sure that's not abused deliberately or inadvertently. There's a whole management layer to make sure that all our customers can play nice in one big shared cluster.

What that then enables is any situation we have large amounts of data, it's kind of the data is heterogeneous or the questions that are being asked of it are heterogeneous. So you can't just define your one clever index and make the problem go away, and where the usage is bursty, which typically I think means it's an internal tool rather than a customer-facing tool.

You look at a big e-commerce site, the engine that drives their shopping cart is used by tens, hundreds of millions of people. It's not bursty. It's just getting pounded on 24 hours a day. But that same e-commerce site, their internal, whether it's log management or business intelligence, like user behavior analysis, whatever, anything that's an internal tool, it's being used by hundreds or thousands of people instead of millions of people. It's going to have a more bursty access pattern.

I think what we've build is interesting across those kinds of workloads, but there's so much that goes into building the whole product and what kinds of data come in and what kinds of visualizations do you want around it and so forth. So we're kind of sticking to our knitting there.

Where we are starting to look at is at other kinds of – Same customer, the engineering team, other kinds of data besides logs that those teams work with, distributed traces and error reports and other things like that. I think our engine is very suited to that and a lot of what we've built in the product in terms of visualization tools and so forth all carries across. So that's something that we're going to be looking at.

[00:53:48] JM: That's cool. So the access patterns for distributed traces and error logs, you think of those as very similar to the log message access pattern.

[00:53:56] SN: Exactly.

[00:53:57] JM: Okay. Well, I know we're near the end of our time. Is there anything else you want to add about the company? I think we've really covered a lot. I know there're certainly more we could have talked about.

[00:54:06] SN: Yeah. Without sort of opening a whole new topic, actually – So one thing I'll just mention briefly, referred a couple of times to dashboards. I just want to mention, there's a whole other side of the backend that we haven't talked about, which is around design for what we call repetitive queries, that's a dashboard, or an alerting rule. Every 60 seconds, check the error rate of this system. Check the latency of that system.

There are these features that lead to the same query being evaluated over and over and over and over again and in huge numbers actually. For example, we have some customers with thousands and thousands of alerting rules. We evaluate each of those once every 60 seconds. So across our entire set of customers it becomes – I don't know, hundreds or probably thousands of queries per second, 24 hours a day.

We have a whole other half of the backend that's built for that, which is basically a time series database, where for each one of those repetitive queries, in database terms, you can think of it as a materialized view. We pre-calculate the result of every one of those repetitive queries and in real time – When we store those in this time series database. In real time as new events are arriving, we evaluate every event against every one of those repetitive queries and incrementally update the time series. There's a decision tree engine we've built that optimizes

those evaluations. So that's another fun piece of technology that we had to build to drive this. That maybe leads to – One thing I'd like to say about Scalyr is it's been really fun for us because the engineering matters. Our customers – We do stuff on the engineering side, the query performance building this other engine for the repetitive queries. We do that interesting engineering work and it actually matters and it does something that the customer cares about. I think that's been a theme of this business from the beginning, is we've found this nice problem domain where the engineering does matter.

[00:55:55] JM: Yeah. That is interesting. That way of architecting it, where you just, along with the ingest process and the transformation to a columnar format, you just say, "Hey, by the way, also check this new log message against any standing queries that correspond to dashboards that might need to be updated and integrate that log message with the records there and then you update the time series database table that corresponds to the dashboard, and viola! You've got a dashboarding solution."

[00:56:29] SN: Yup.

[00:56:30] JM: Steve, I want to thank you for coming back on the show. We talked about a lot and I think there's a lot more we could have covered. We didn't even get as far as the business side of things, competing with cloud provider products, because we did a show recently about Google's stack driver thing. I don't know. It's interesting that you're competing with the LightSteps and the Datadogs and then also the major cloud providers now. So the competitive environment is also quite interesting, but obviously not the complete focus of the show.

[00:56:55] SN: Yeah, interesting is a good word though. Yeah. It does keep up on our toes.

[00:56:58] JM: Cool. Steve, thanks for coming on the show, and I look forward to having you on again sometime in the future.

[00:57:02] SN: Okay. Thank you. I look forward to it.

[END OF INTERVIEW]

[00:57:08] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance.

The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

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

[END]