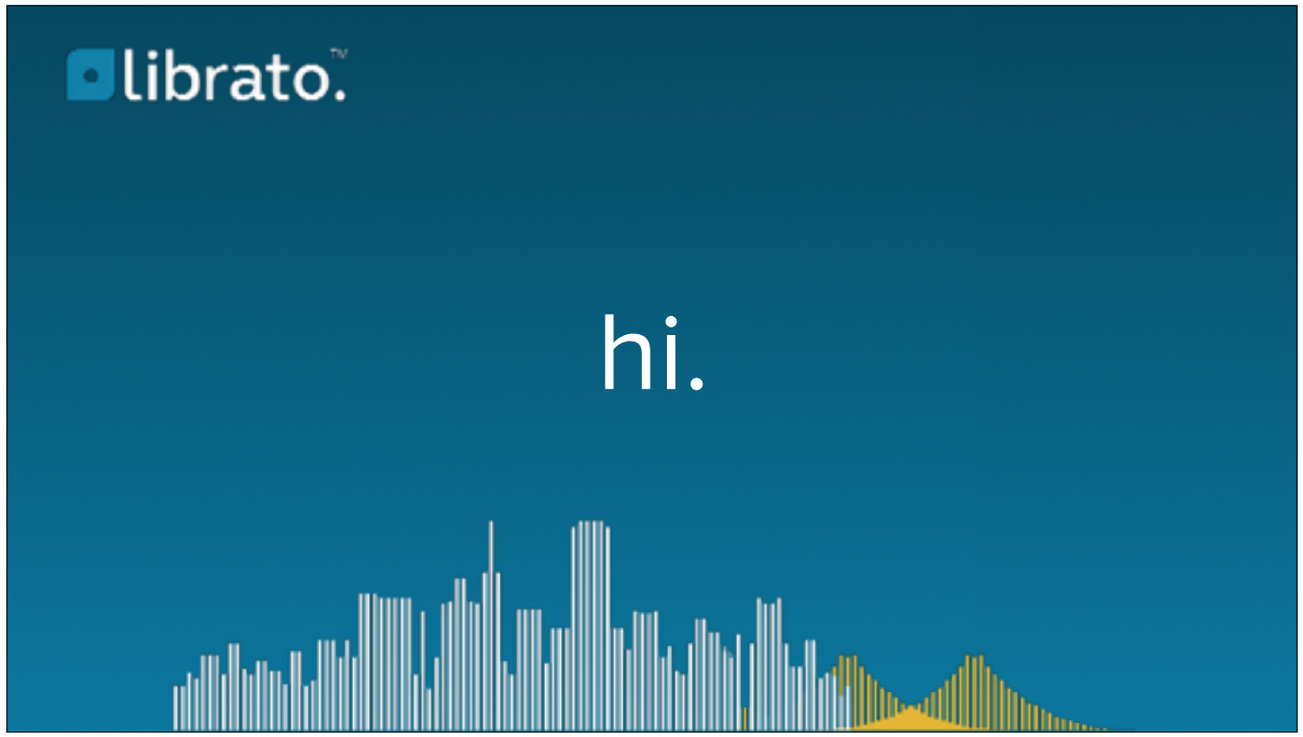


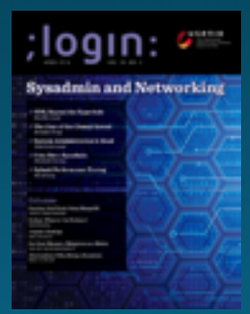
Hi everybody

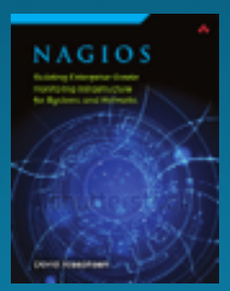
My name is Dave josephsen, I’m the developer evangelist at Librato, and today I want to talk to you about Monitoring, which is a subject that I’ve spent a lot of cycles on personally

librato.

hi.

dave@librato.com
@davejosephsen
github: djosephsen

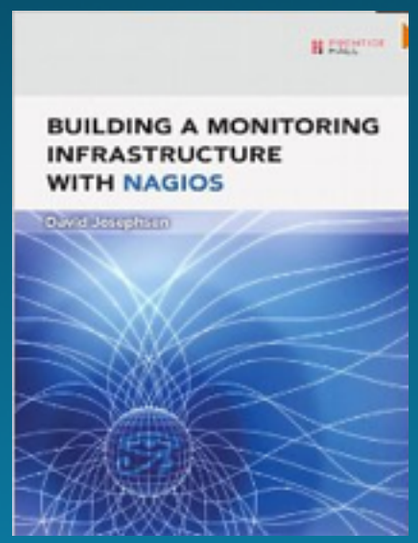




In fact you may know me as the worlds foremost authority on having written technical non-fiction books about monitoring that have

squiggly things on the cover. It is a distinction in which I take great pride

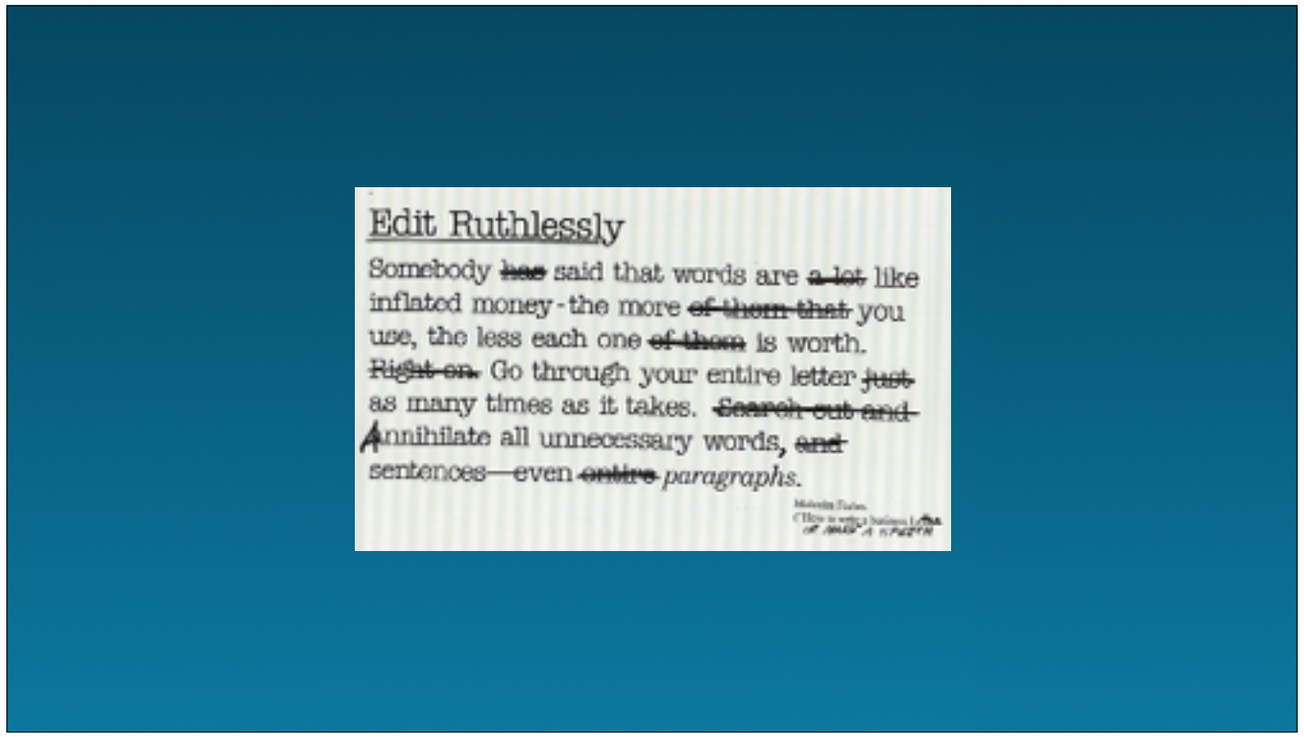
librato.™



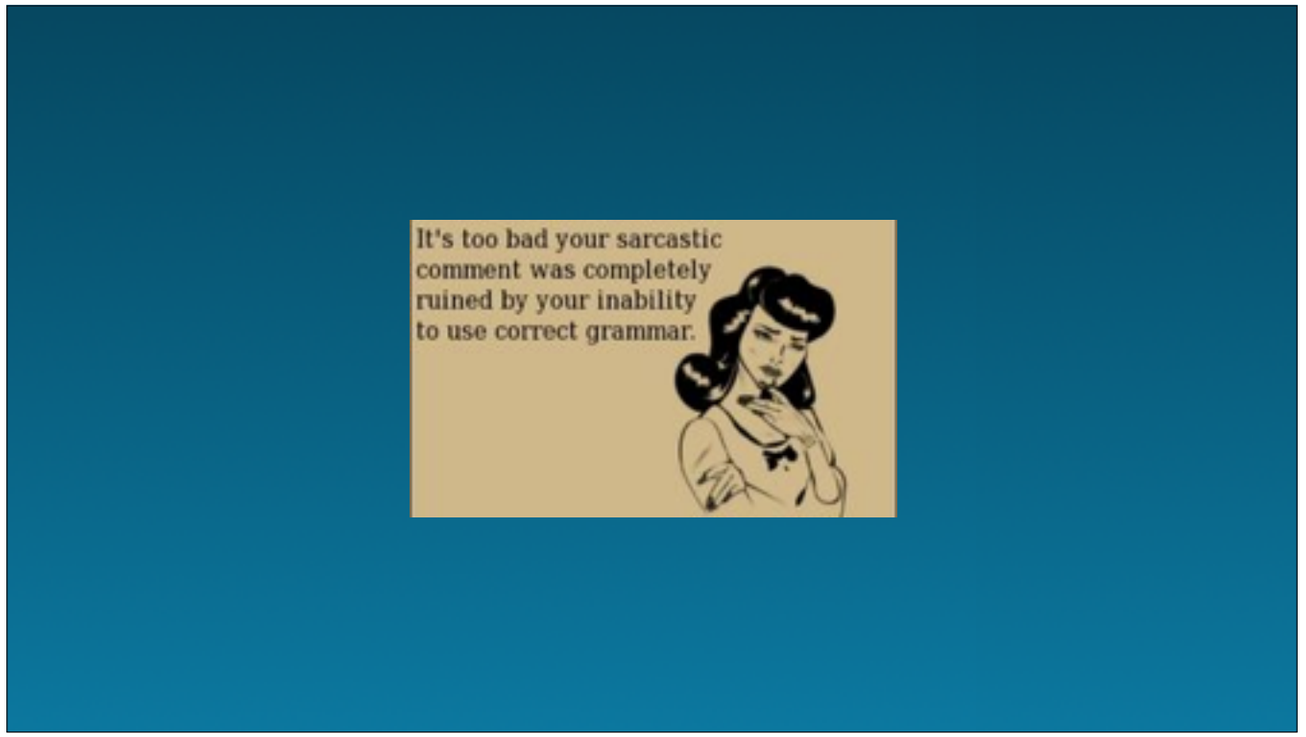
2007

I wrote my first book on nagios nearly a decade ago now, like I’m seriously milking it at this point. And It's fair to say at the time I wrote this, I had no idea what I was getting into. The younger me who wrote this was not a book writer, he was a barely sentence writer; and anyway it quickly became apparent to me and my editors

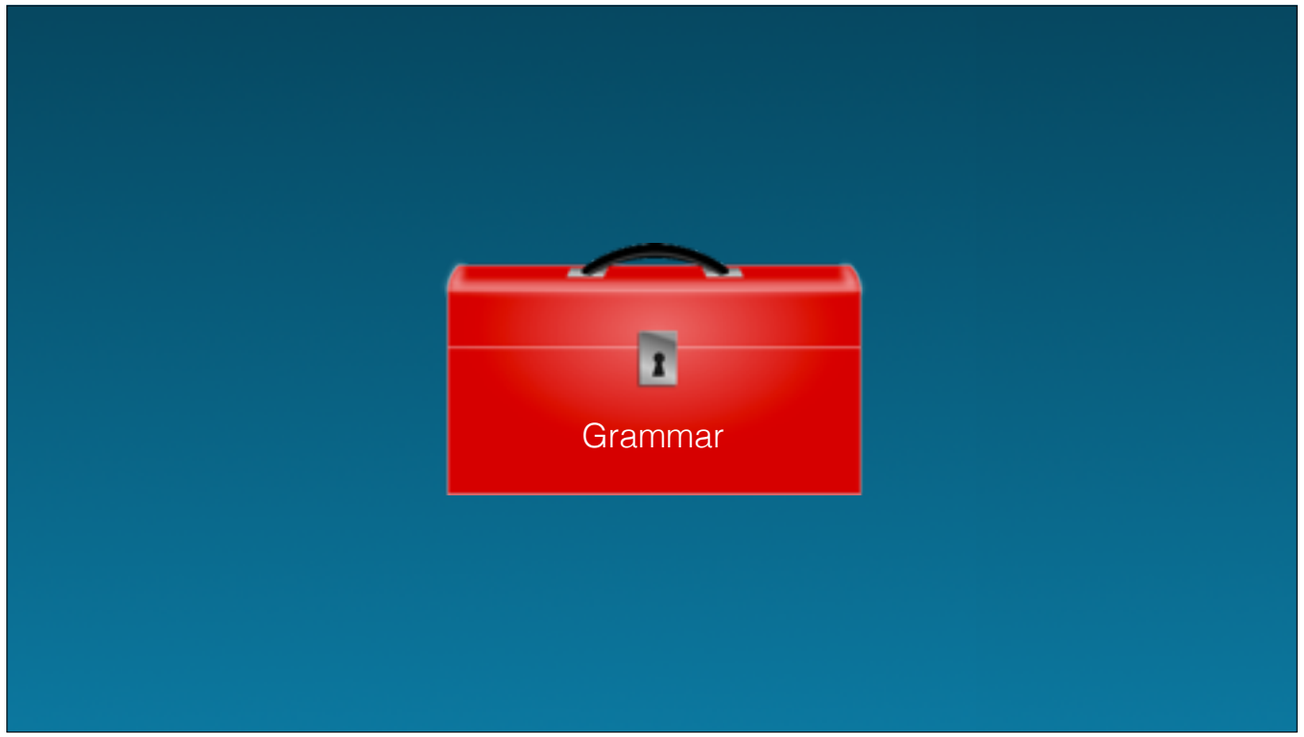
that I hated editors. And really the entire editorial process. I hated their tireless meddling, how they would destroy what I thought at the time was nuance, but what actually just turned out to be me repeating myself. But yeah they would delete words, like they would literally steal words from me.



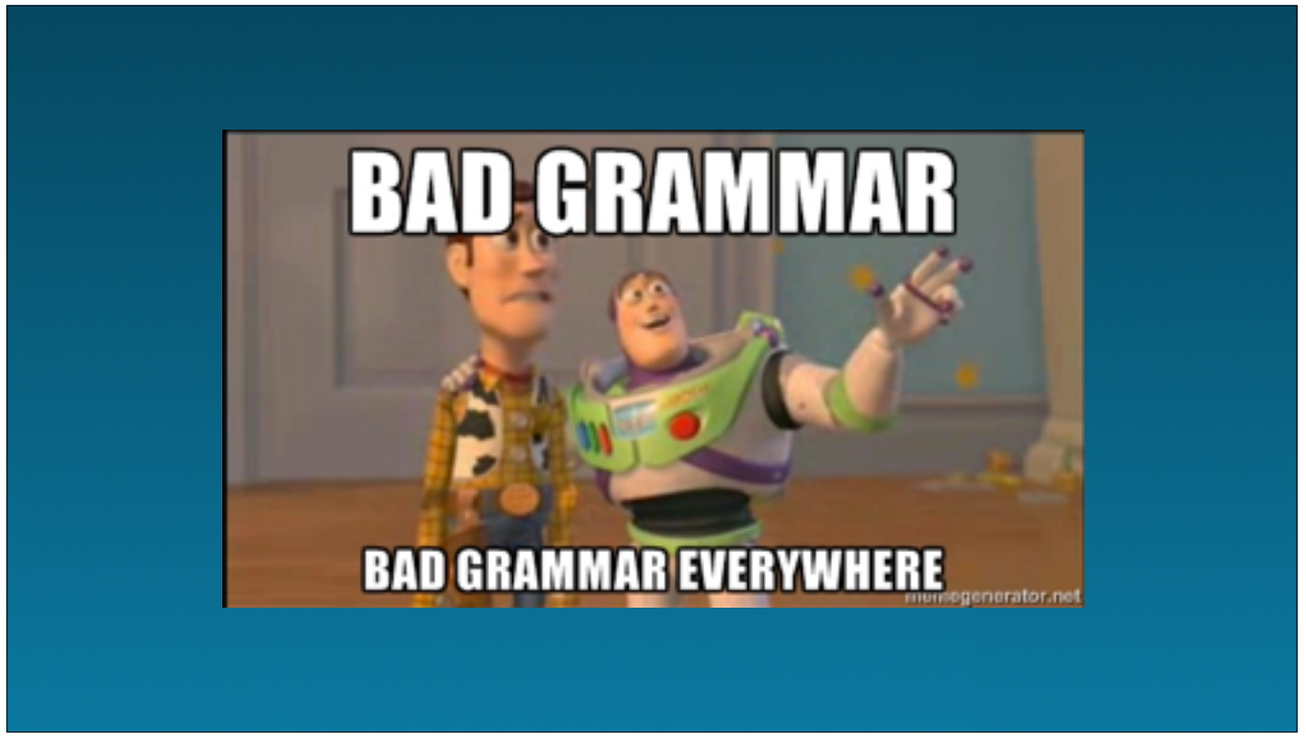
They would take my writing and give it back covered in annoying markup and I've seen their judgy pintrest boards, I knew that they were having fun doing this to me. So I would stomp and rage and argue with them about every little mark they made.



But it was hard because they knew all of these rules of grammar and parts of speech; like they had this entire vernacular they could bring to bear to make me look silly. So I decided that I was going to learn grammar because I already knew that my every sentence was perfect, I just needed to know the jargon that describe why I was right, and grammar would give me that



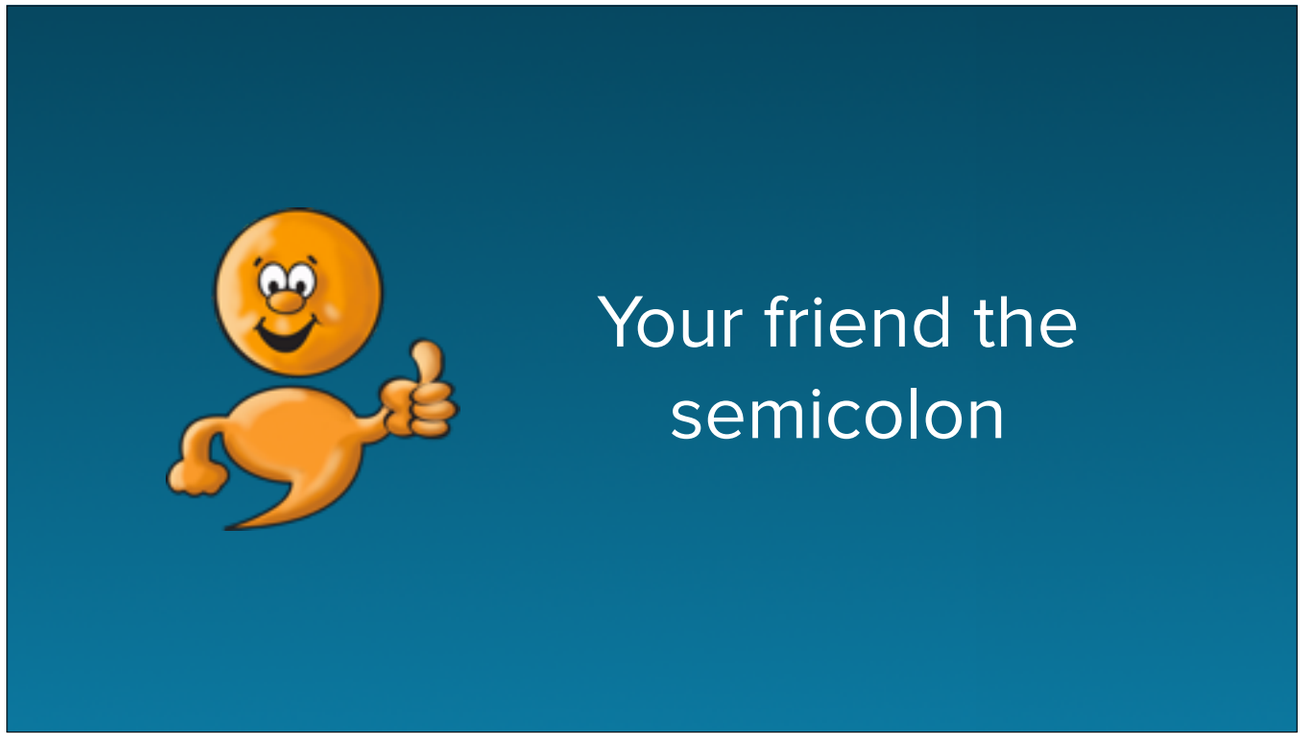
But I pretty quickly discovered that grammar was not a set rules, or at least not just a set of rules; it was more like a set of tools. And the more I learned about the tools, the more it became obvious



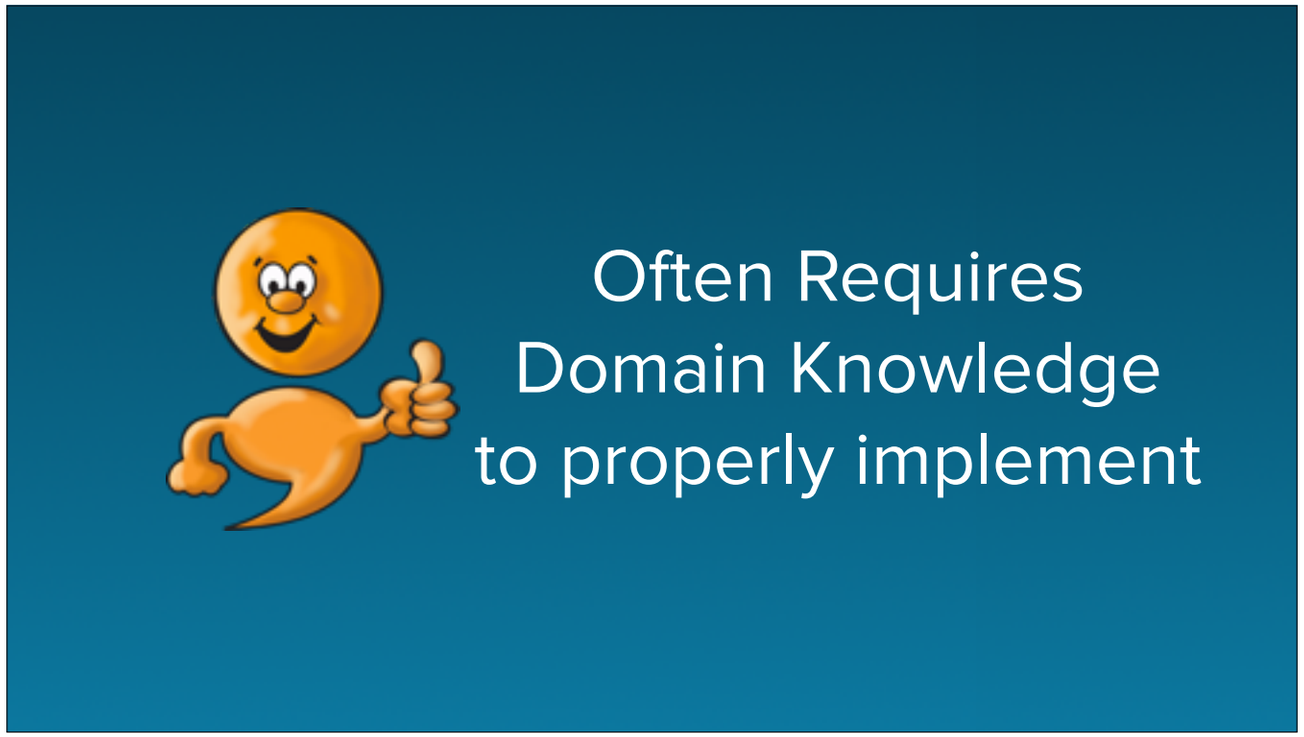
how awful my writing was. And it wasn't just my writing, it was my thinking; like, my ability to adequately formulate and express a thought had been impaired by my incomplete knowledge of these tools.

The more I learned, the more I began to rely on my editors. Even when they were outright factually incorrect, every mark they made invariably pointed out something I now wanted to change

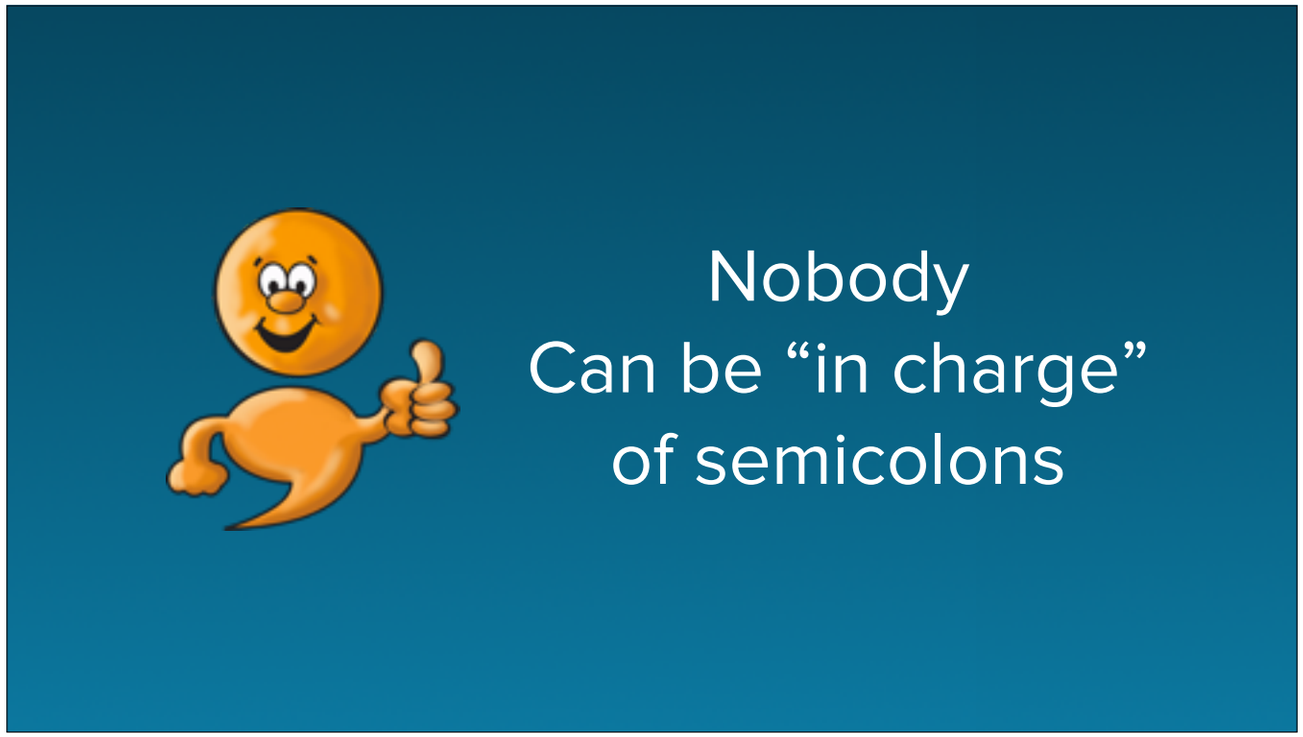
So learning this common toolset transformed my relationship with these people overnight into something much more conversational. They went from people who gave me headaches to people who gave me feedback. And the only thing that had changed was me.



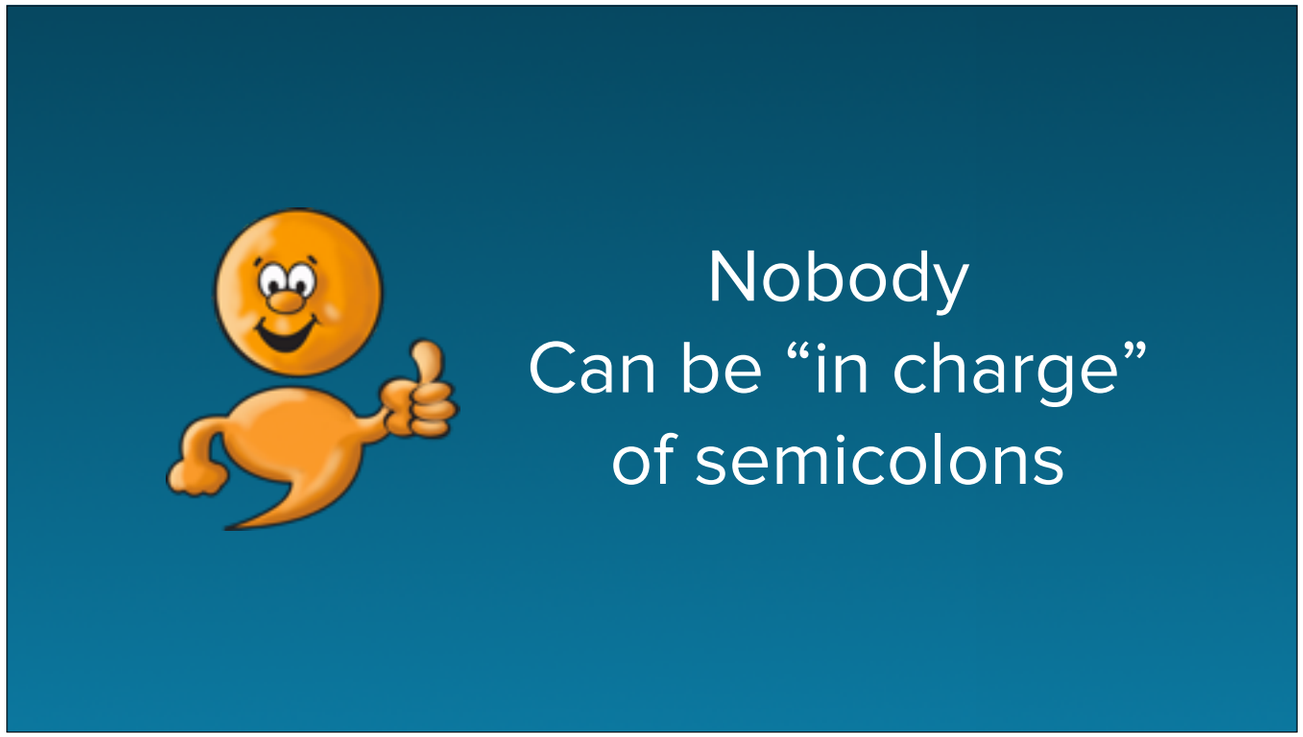
One of my favorite tools in the grammar toolbox is the semicolon. Semicolon's are actually crazy powerful, and if you don't know how to use them, then you have a severely limited ability to imply a relationship between two ostensibly unrelated ideas.



The semicolon is a pretty decent metaphor for monitoring. Everyone knows about them, but fewer people know how to use them properly. And properly using them often requires domain knowledge. Like you need to know the meaning of the prose to properly use a semicolon



So nobody can be in charge of semicolons. We can’t have one person sprinkle a bunch of semi-colons down, and then get another person to fill in some words, and expect to get something meaningful out of that. We need to work with our editors using a common grammar



And that’s also true of effective monitoring. We need to work with our devs, and reason about what we’re creating, and use monitoring as part of a common grammar for implementing our vision. And sometimes we’re going to need domain knowledge to do that.

In fact, thats true whenever we create something with other people, be it a tech-book or a micro services infrastructure. We never say, hey, yall wanna go grammar for a while? Nah I already grammard today. like, we can't make a verb of grammar, because grammar outside the context of a creation is meaningless.

And really the point I want to get across to you today, is that monitoring is the say way. It doesn’t make sense as a verb outside the context of a creation. At the time younger me wrote this book, I was in charge of monitoring, which is exactly like being in charge of semi-colons.

Monitoring is also just a tool in the grammar of web operations engineering, in the same way the semicolon is a tool in the grammar of english. Everyone has to know how to use it, and it ONLY makes sense in the context of what we're making together.

Uptime is a symptom. (not the goal)

Of course, younger me thought that "uptime" *was a creation* a thing to be made, an engineering endeavor; which, is a commonly held belief that got us through the first decade or so of the 21st centry, but it isn't really true. Uptime is a marvelous thing, but it's never the actual goal. The actual goal is always selling widgets or running services that help people, or landing robots on mars or you know.. human endeavor; that sort of thing.

Uptime is a symptom. (not the goal)

But that was lost on younger me. He was obsessed with uptime. My job was to stand-up systems, and run infrastructure services, and to make sure all of it remained available,

Uptime is a symptom. (not the goal)

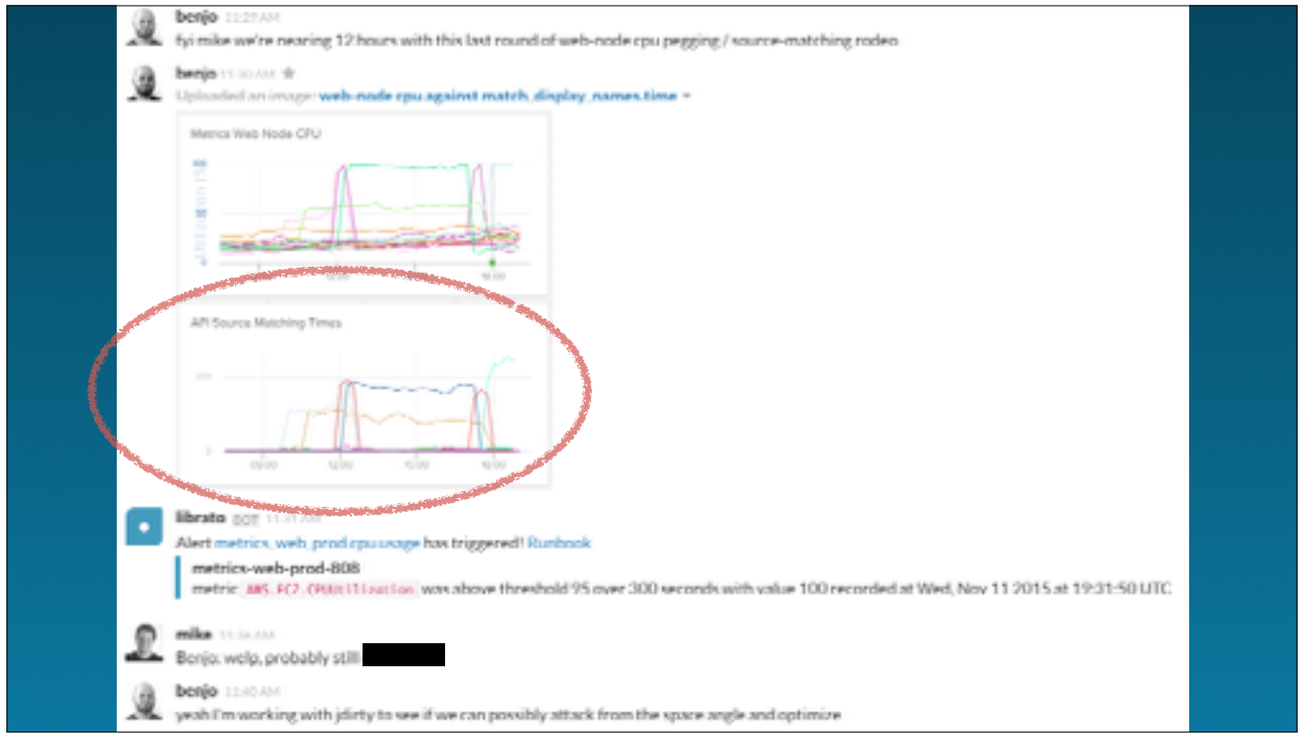
I had no desire to reason about what was happening inside the probably horrible applications we were developing, so putting younger me in charge of monitoring basically gave you uptime data, because that's all I was interested in.

lemmeshowyousomething

Let me show you something..

This is a snippet I took out of a conversation in the Librato Ops channel. yesterday morning. It's not special, we have conversations like this one all the time.

And what's happening here is that Ben who is an ops guy, is tracking a correlation between Web-Node CPU utilization up here in this top graph



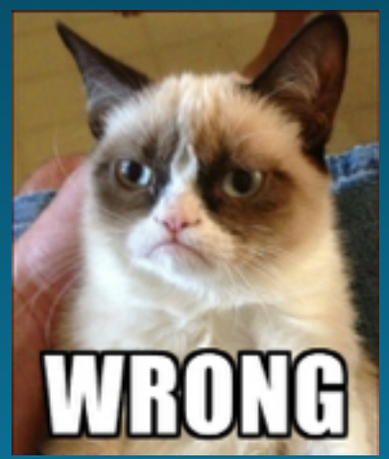
and High cardinality of sources per metric on this bottom one. Basically, some user has lots of sources for the same metric, which is causing our API to spend time and effort mapping them all

And Mike who is a dev basically acks this like, welp.. yeah that sucks. We know why, and we can spin up instances if this becomes a problem, and that's fine until the roadmap catches up to this behavior. So this is dev and ops having adhoc conversations about emergent behavior in our applications. Before I joined Librato I'd never been able to have conversations like this one.

I mean, think about the domain knowledge ben needed to make that correlation. Even if he's come across this before, he has real domain knowledge about this application. He knows how it reacts to metrics with a high-cardinality of sources, and he can use that knowledge to take action. Monitoring expert me never had this.

- Makes money
- Can Therefore Spend Money
- Buys the easiest APM it can afford

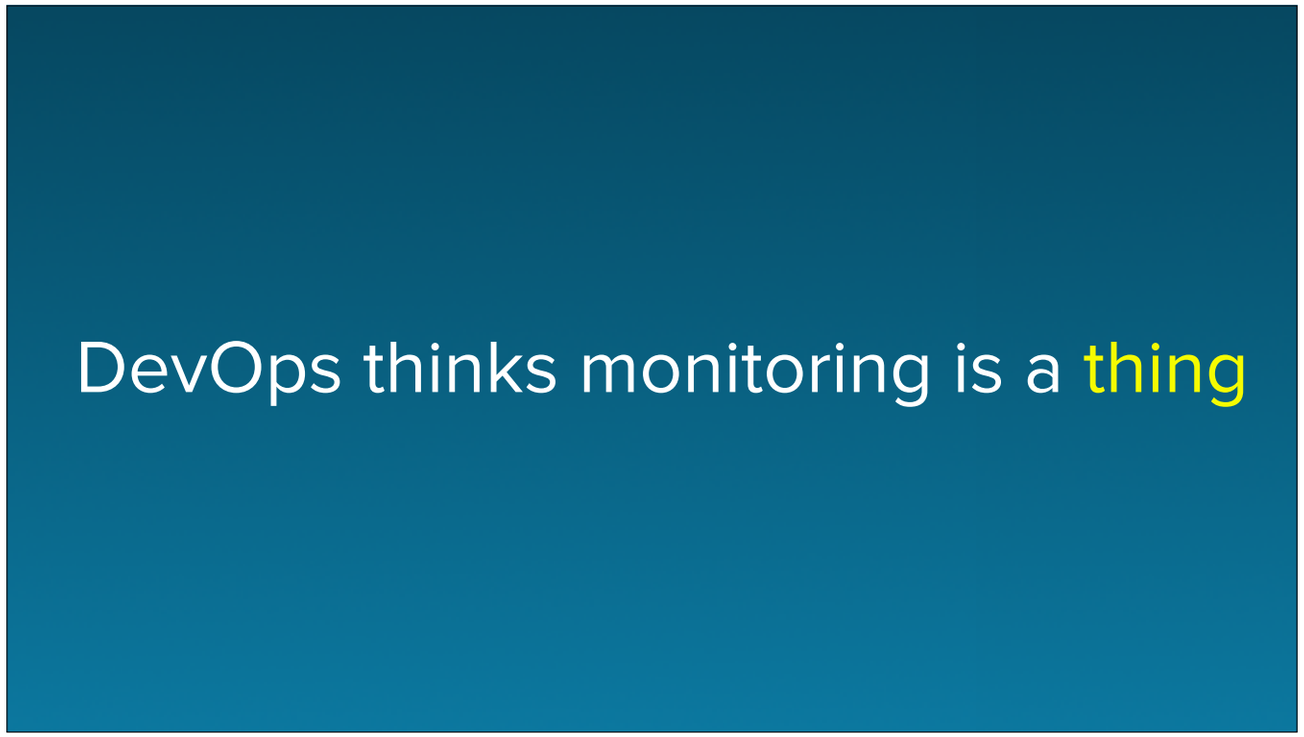
Because my developers always operated in a completely separate sphere of reality. They MADE things. And those things made money. And therefore they could spend money, so when they wanted to understand what was going on in their stuff, they'd spend money on APM tools.



- Does not make money
- Cannot spend any
- Deals with it.
- Hates you.

Meanwhile I was a cost center; literally a burden, so when I wanted to know what was happening in my stuff I would spend headaches wiring together free stuff.

So I was looking over here and they were looking over there, so really it never even occurred to us to talk. We believed these were separate undertakings, and therefore we had no common grammar that allowed us to bridge this gap



And devops never fixed this for me. At best devops merely created more categories of monitoring that more individuals could be in charge of. Devops acknowledges monitoring as an activity, which again, it is not an activity any more than grammar is an activity.

lemmeshowyousomething

Let me show you something..

here, Ray who is a devsy type person, says hmm.. spike in service X, and then benjo, who again, is an opsy type person, agrees with him; adding that it seems to be still getting worse; so this needs to be looked at, before it results in other things going sideways.

That first observation - that this problem appears to be effecting subsetX — is more or less objective; that was in the alert. But the second observation -- that this problem is a sustained thing -- is more subjective, and so, Benjo includes a link to the telemetry data he's looking at, in this moment in time, because if you’re literally staring at data, why attempt to describe it when you can just show me the data right?

And then there's a whole bunch of stuff that doesn't need to be said because It's right there, a picture worth 1000 words and all that. We know it hasn't peaked. We know where it's at now, and how long it's been growing for, and etc.. Monitoring has become a common basis for these guys to understand this problem.

See where I’m headed here? Engineering is a conversation just like the editorial process is a conversation. We're having a conversation about this creation whether it's a book, or a microservices infrastructure. We can specialize; we can be devsy or opsy, writey or editory, we can take responsibility for this part or that part, but for the whole thing to be great we also have to be interdisciplinary enough to be able to talk to each other about it, and in order to be able to talk, we need grammatical tools. This that you’re looking at here is a grammatical tool.

You can't put one person in charge of the monitoring toolchain, and expect to have conversations like this one. You won't. Younger me didn't understand this, and therefore it was impossible for younger me to have conversations like this one that Benjo and Ray are having here.

like, I couldn't even draw this graph. Not because I lacked this or that piece of software, or this or that vendors product, but because I had no means of understanding and instrumenting devsy stuff and they had no means of helping me understand those things. So I stomped and raged at my devs, just like I stomped and raged at my editors.

lemmeshowyousomething

Let me show you something else..

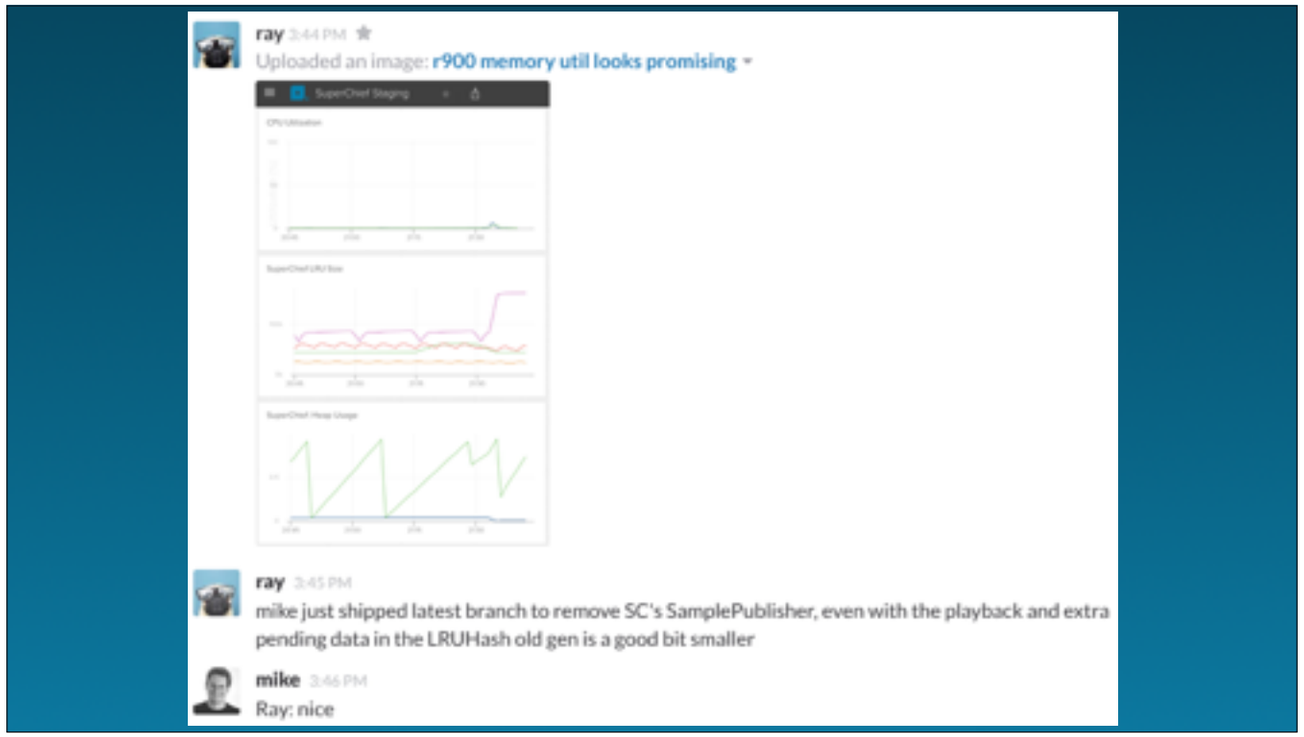
This is Mike and Peter. Different dev, Different op, basically the same issue; a cardinality spike in the same exact service, and look at the conversation. Pretty much also exactly the same. We saw a spike, we looked into it, subjective claim is made, subjective claim is backed up with data. Everyone on the same page. So when we treat monitoring like grammar, we change how the people interact to the extend that we can literally swap out the people and still get the same result.

And everybody who knows this grammar can have all kinds of these really useful conversations because although there is specialization, the grammar bridges those disciplines. Semicolons belong to everyone.

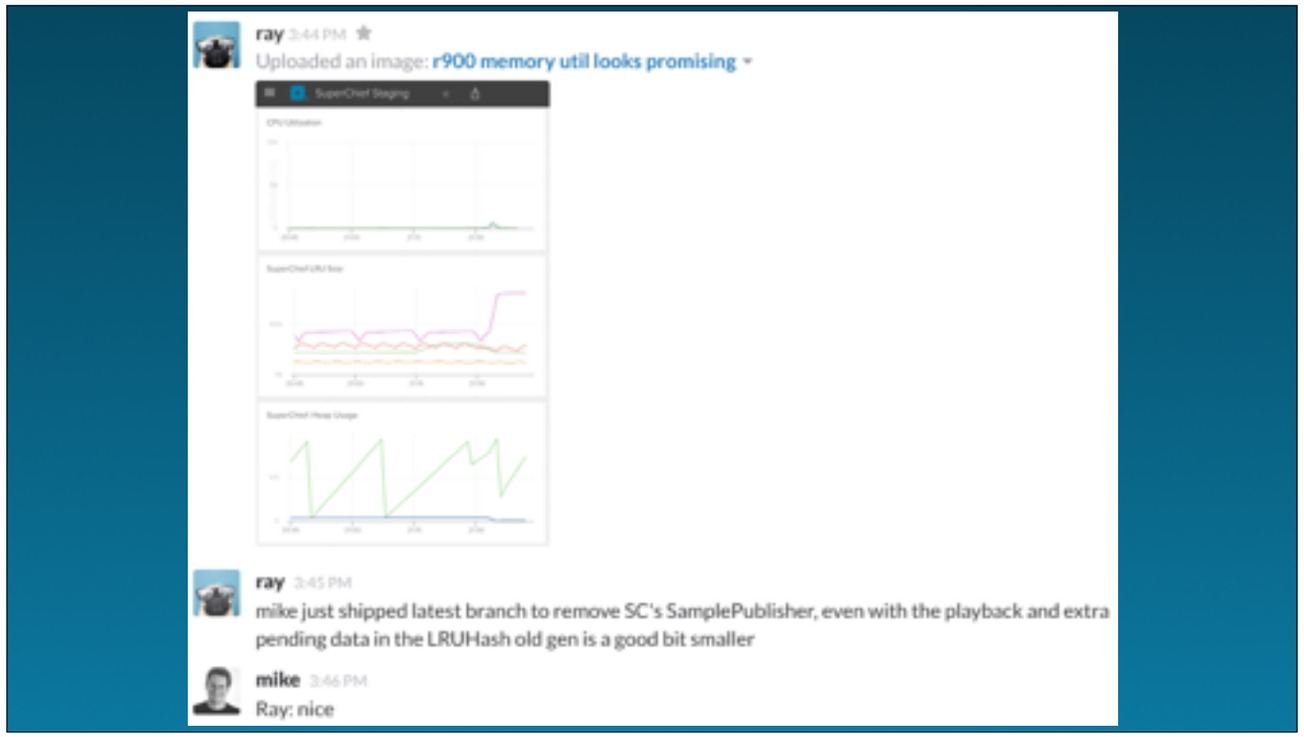
Dev can say you know, based on what I know about how this service works, I bet this problem is causing a drop in metric X

And Ops can go put their hands on the data they need to confirm that and share it like hey look you were right.

and dev can be like yup.



or someone can say hey, I just pushed a change that should improve the memory management of this service, and then watch the result and discuss it.



Nobody is in charge of semicolons here, and yet everyone understands how to use them. Just like with me and my editors, this grammar has enabled a cooperative and conversational working relationship.

lemmeshowyousomething

Let me show you something else..

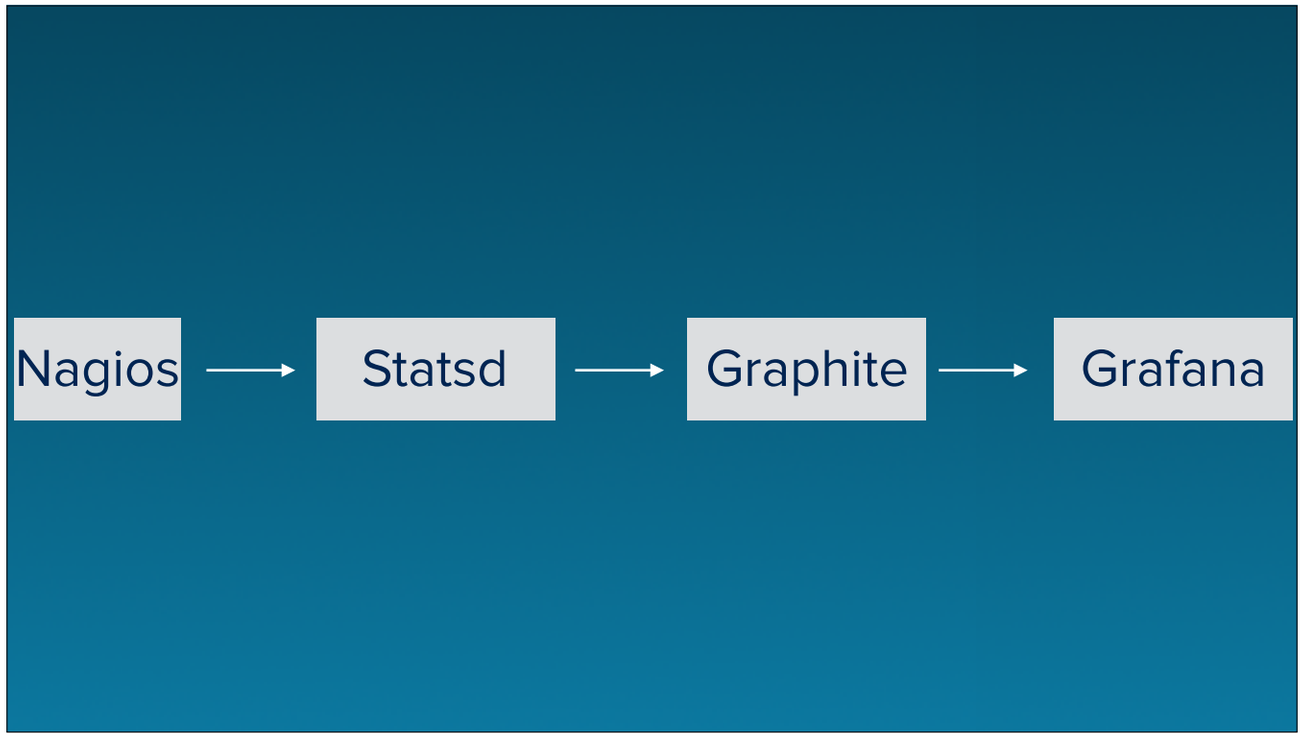
So in this snippet, Jared and Mike, two devsy people, are trying to figure out a problem with this service that we run internally. It's basically a service that enables a user to take some number of metrics streams, and combine them using math.

And it's not an easy service to run by the way, This is the sort of thing that it's easy to DOS yourself with, like dear user, here's literally an interface you can use to make our servers ad-hoc compute arbitrarily complex time-series workloads, using as many data points as you've stored in the last year. have fun with those type globs.

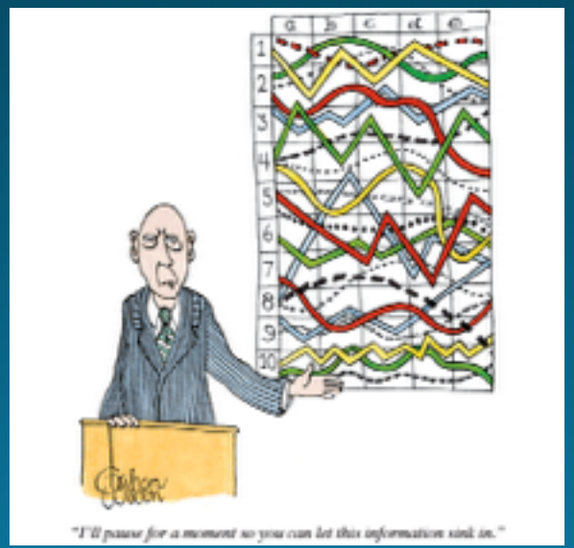
so a lot of tuning work goes in to a service like this. It's difficult to reason about where the bottlenecks are going to be, and it's difficult to predict how your users are going to break it. So to do this well, we have to measure things like how long it takes for this service to render this or that type of computation, which is what we're visualizing here.

And of course, this was another impossible feat to achieve for younger me. Because this is hard-core domain knowledge. You need to have built this service to know that you need to monitor this, and you need to take those measurements from inside the application while it's running in production. To draw this line you need tools expertise and domain knowledge, and I only had the former, and everyone with the latter were yucky microsoft using people who didn't want to talk to me any more than I wanted to talk to them, and we all suffered as a result.

but the point is, if you think — like I thought — that monitoring stands on it's own, then you won't ever draw lines like this, and therefore your ability to run services like this one will suffer.

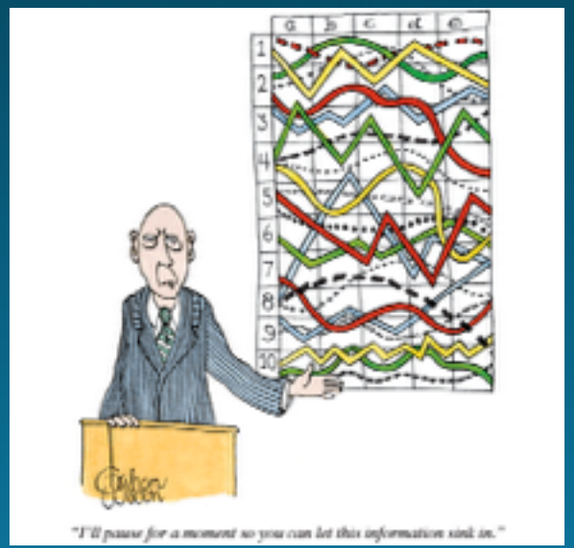


You'll buy some software, or you'll wire nagios to graphios to statsd to graphite, and make some CPU graphs, and call it a day. And if something goes wrong you'll rub your chin and stare at the CPU graphs because that's what you have to stare at.



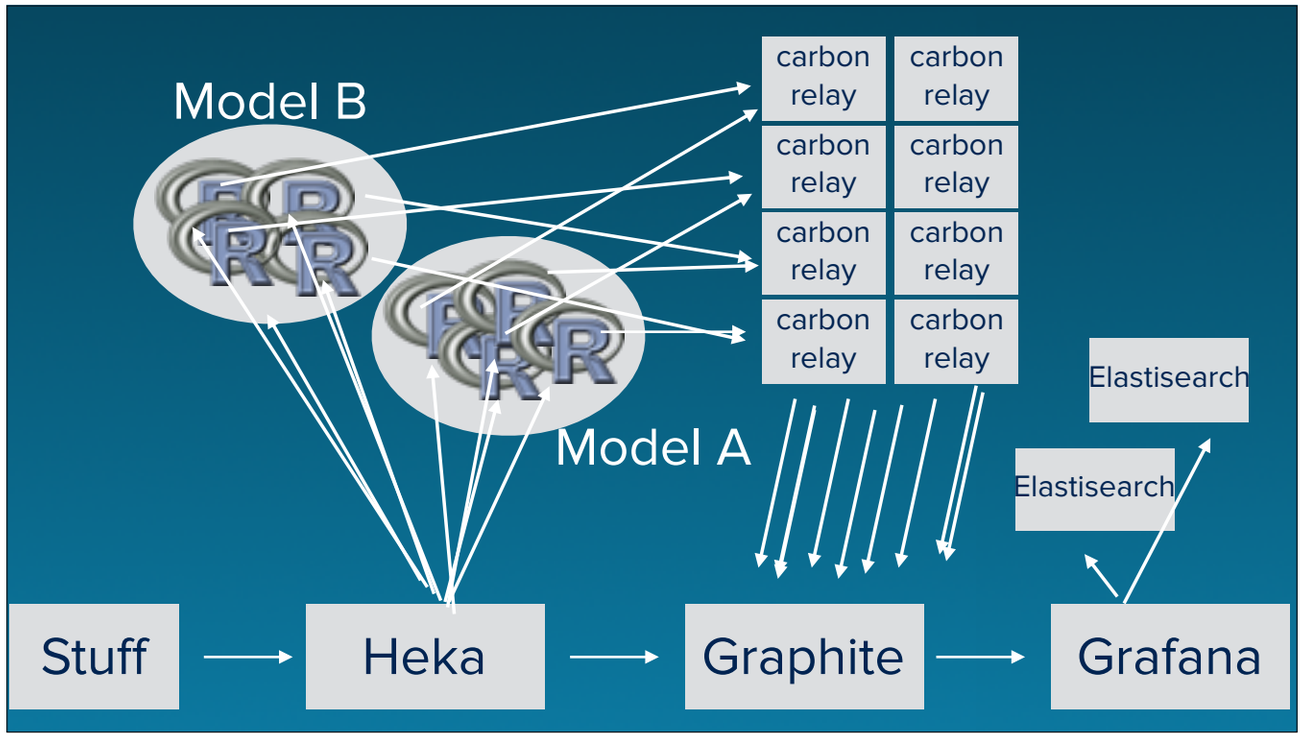
Clearly what we need are more metrics

And when those CPU graphs don't really provide any insight, maybe you'll buy into the fallacy that you need better tools to provide you more data. You'll choose tool A versus tool B because tool A has 4000 metrics, and tool B has 500 metrics.

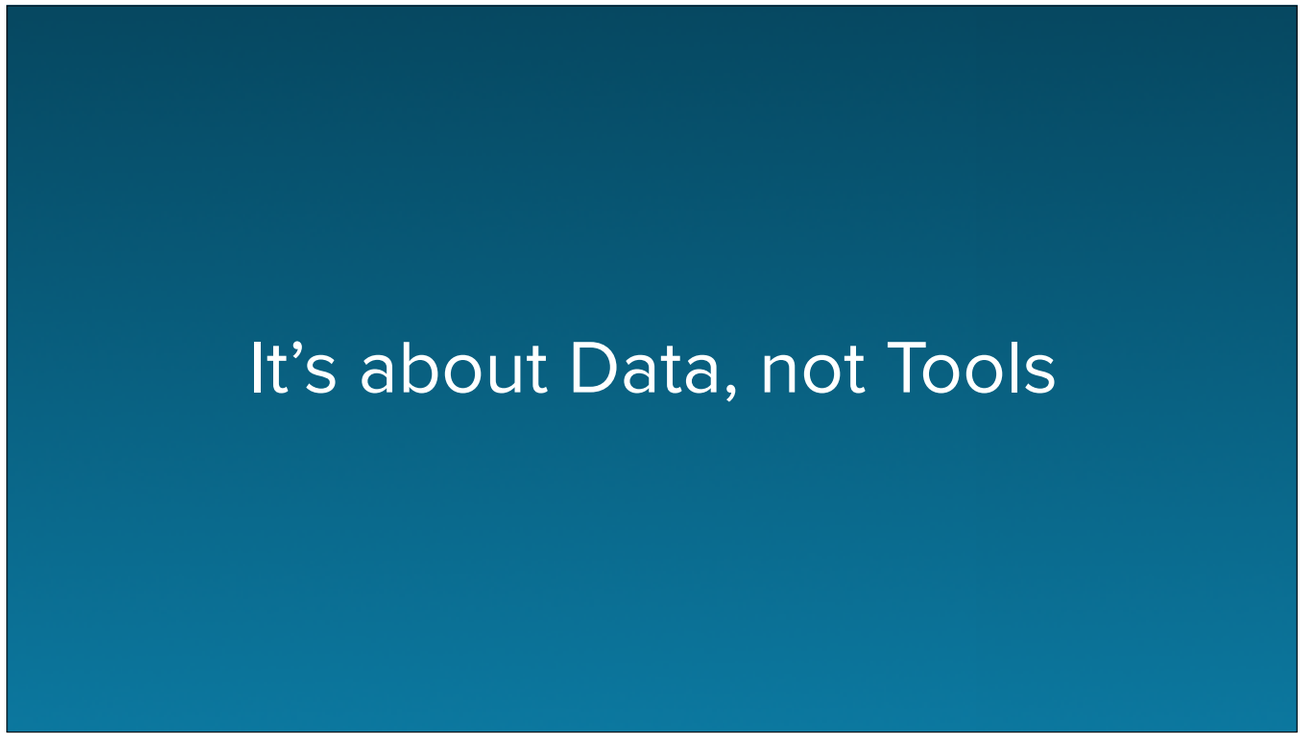


Clearly what we need are more metrics

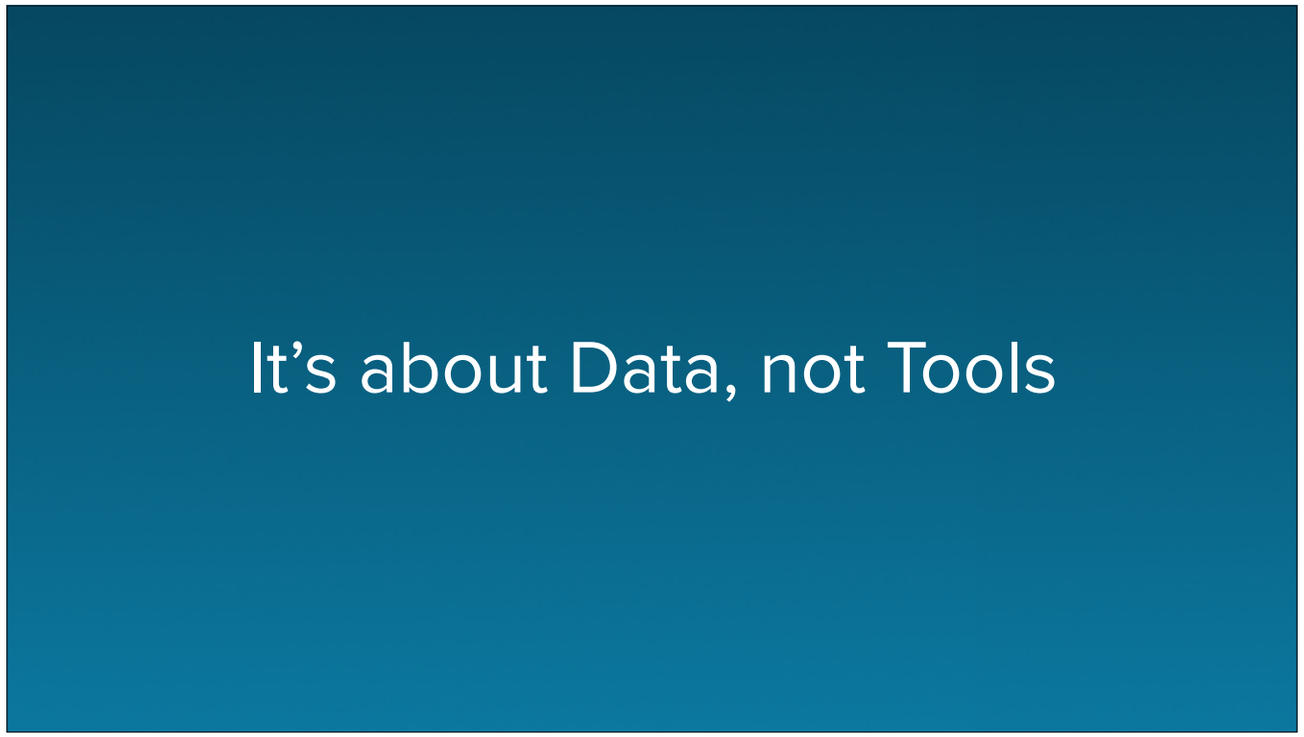
Or maybe having already been stuck with tool B because their salesman had an awesome scottish accent and took your boss golfing, maybe you'll bolt on tools C and D because yay MORE metrics.



And soon you'll have 17 different data collectors wired up to send a million metrics to 5 different metrics processing systems, and you won't bother to look at any of it when there's a real problem, because you won't understand how the data is derived, or what it's actually measuring.



Like here’s some thought leading for you. For a metric to be useful to you, you need to know what it means. How often is that collector polling? What precisely is the collector measuring? How is it being summarized? It’s not impossible to make sense of 4000 metrics that someone else has chosen for you, but I mean you really need to dig in, and understand what all of those names mean, and what their collectors actually measure



And at the end of the day you shouldn’t be that invested in monitoring. You should be invested in results. Don’t choose tools, choose data. Then use the tools you need to get the data. Make Data the first class citizen.

Have you seen this? I'm guessing you have. Younger me was in love with the spirit of this sentiment. YES. Measure everything.

I no longer have any idea what his means. I mean I kind of get it; we live in this weird universe. There is no now. No quantum unit of time that we can measure as being now.

Don’t Measure Everything

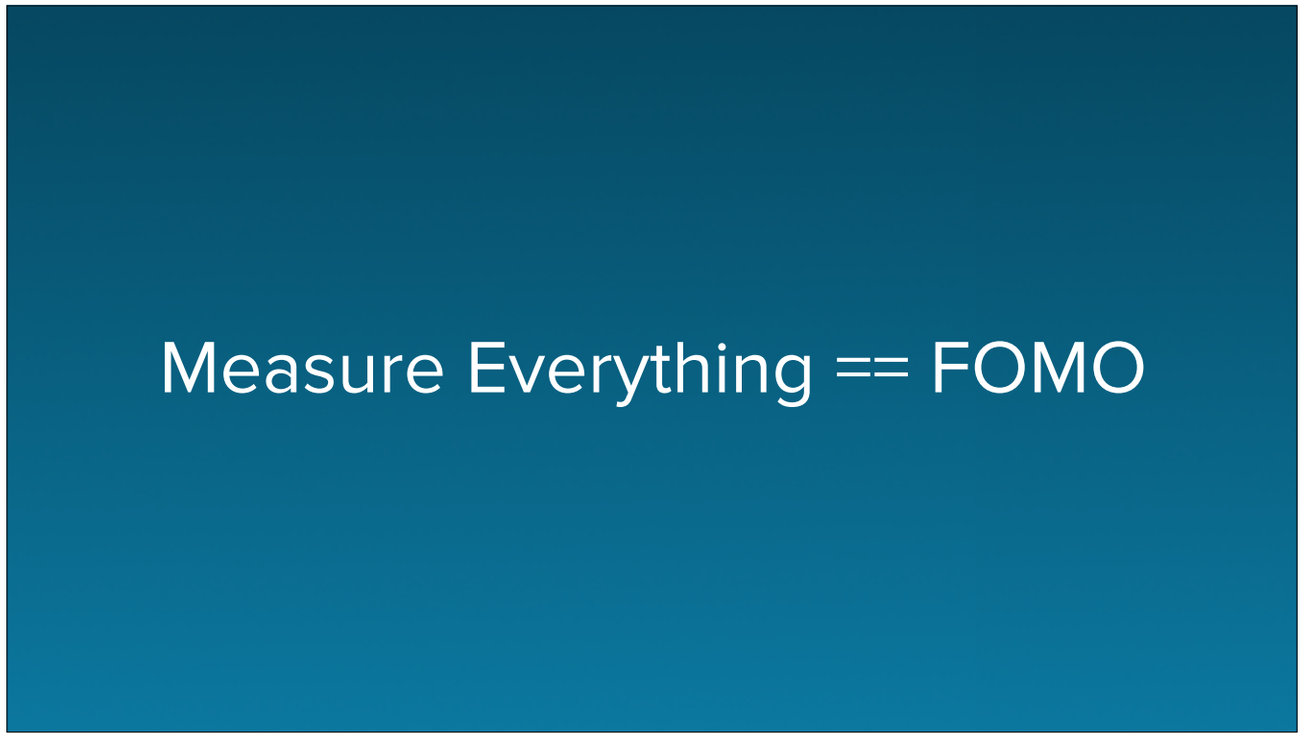
Somehow we just exist on this infinitesimal line that we can't measure between stuff that hasn’t happened yet, and stuff that’s already happened.. NOW that's history. Was that now ever now?

Don’t Measure Everything

Stuff never actually happens. We just live in this perpetual state of new stuff that already happened.

And just NOW all of this stuff happened inside our computers, some of it was logged, most of it wasn't. And now it's history and there's no way to get it back. That happened. It's lost forever. Escaped into the past unwitnessed.

How many millions of things just happened inside our thousands of computers that are now lost to history? Which of them was important? We can't know, we have to measure everything, or we won't have it later when we need to figure out what happened.

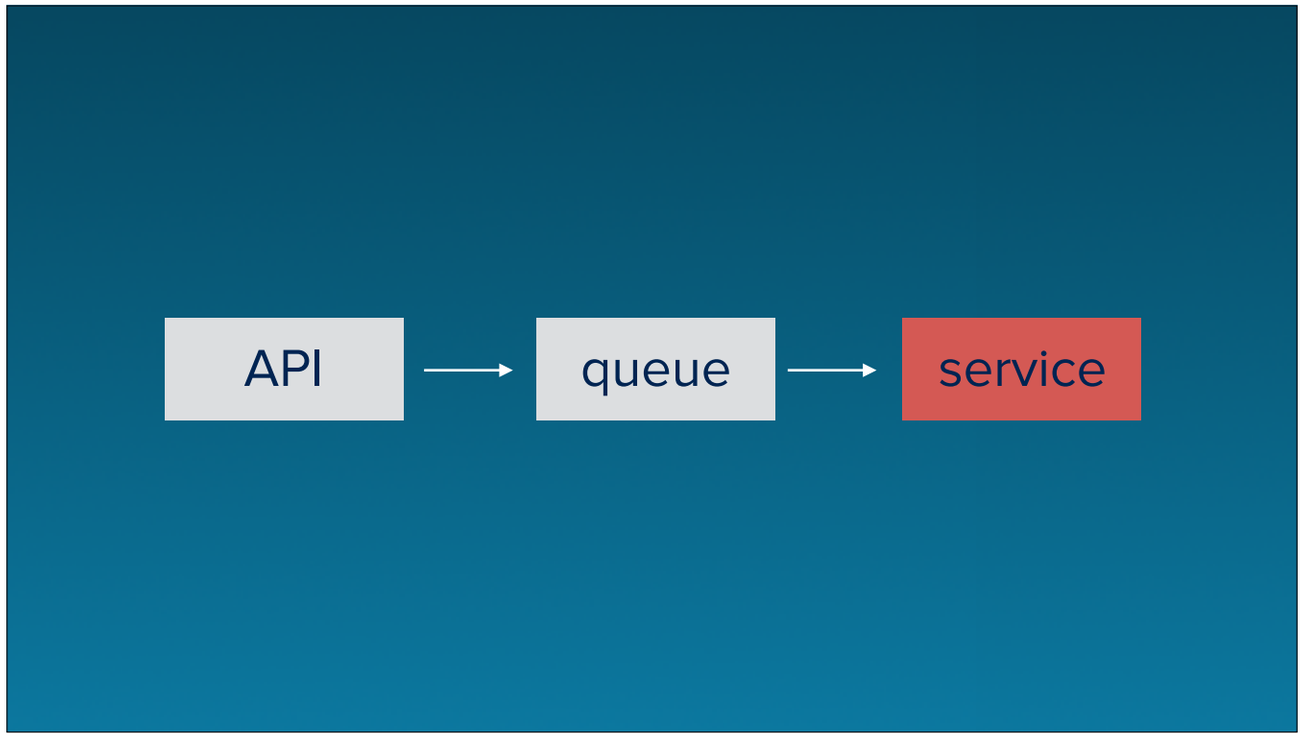


This is FOMO; like what you’re describing to me here is probably the ultimate expression of Fear of missing out. Even if measure everything were possible which it isn’t, you’d never be able find anything in that huge pile of everything.

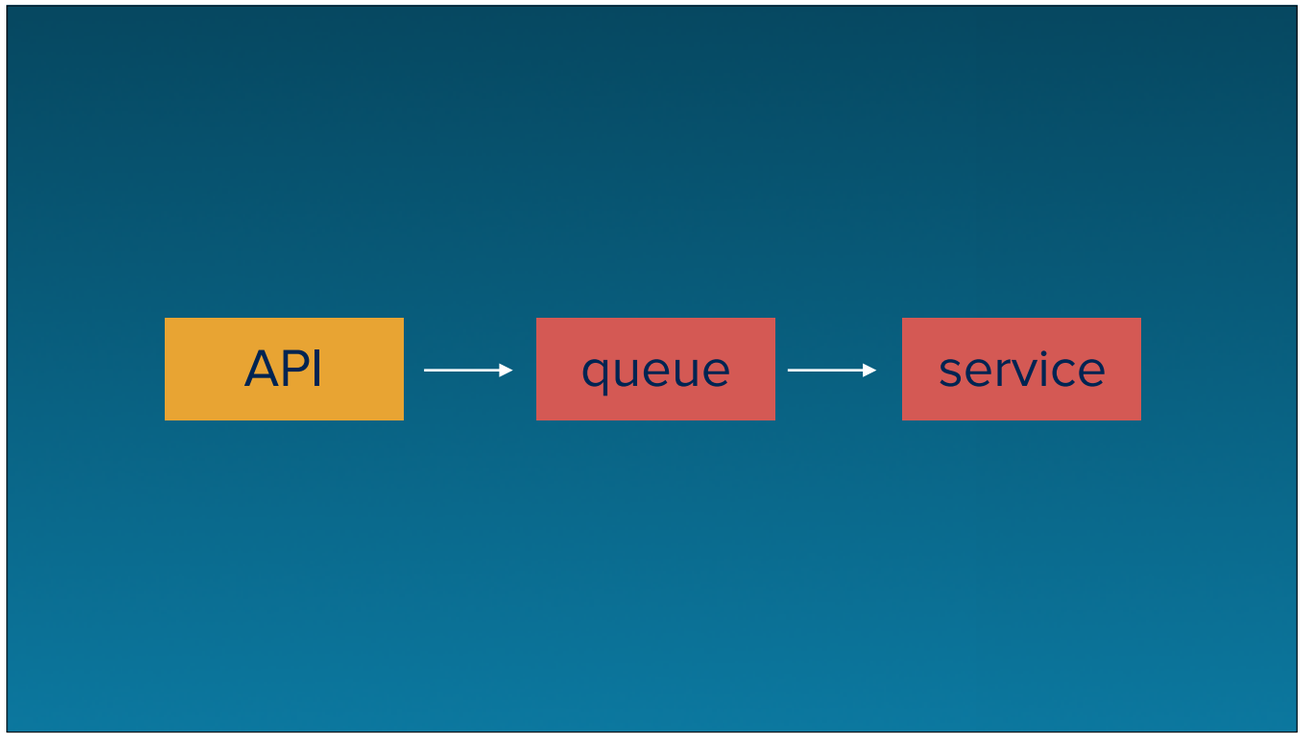
lemmeshowyousomething

Let me show you something

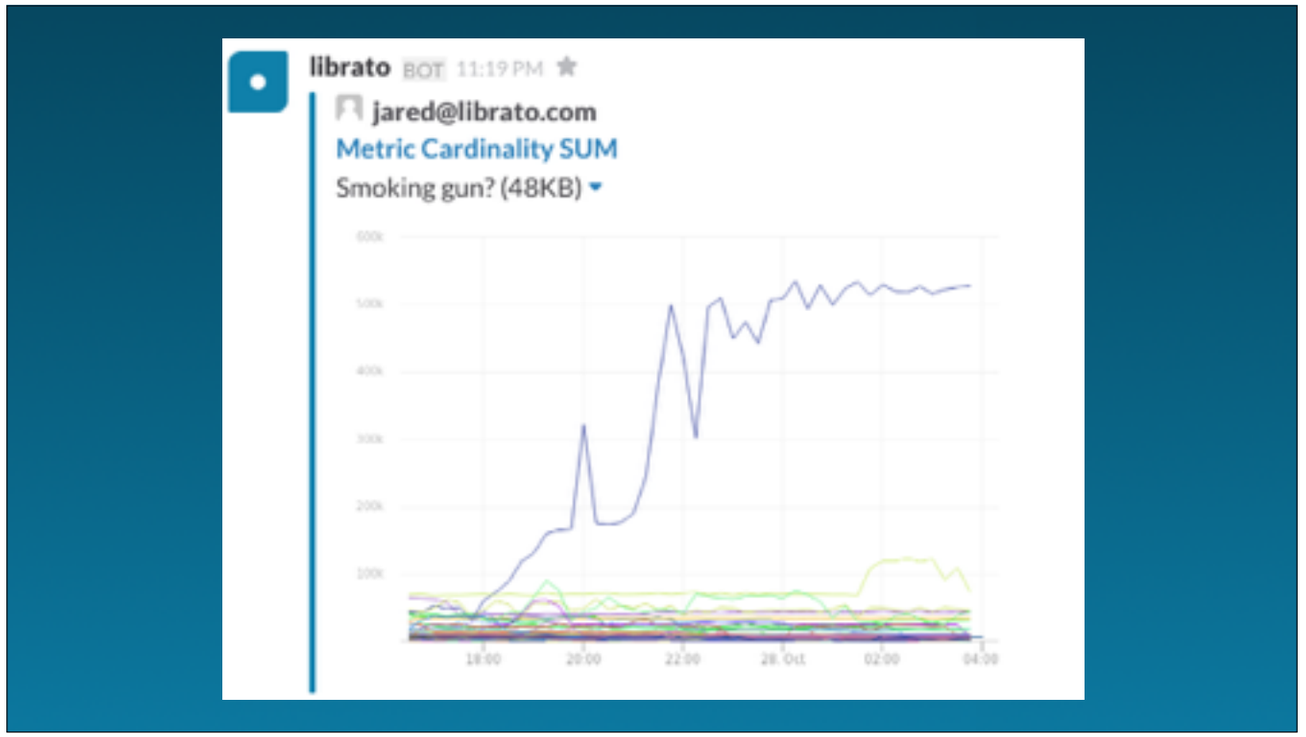
So here’s mike and jared again. They're troubleshooting that same “do math for me”service, and this time it's gumming up the works, taking too long to respond to requests.



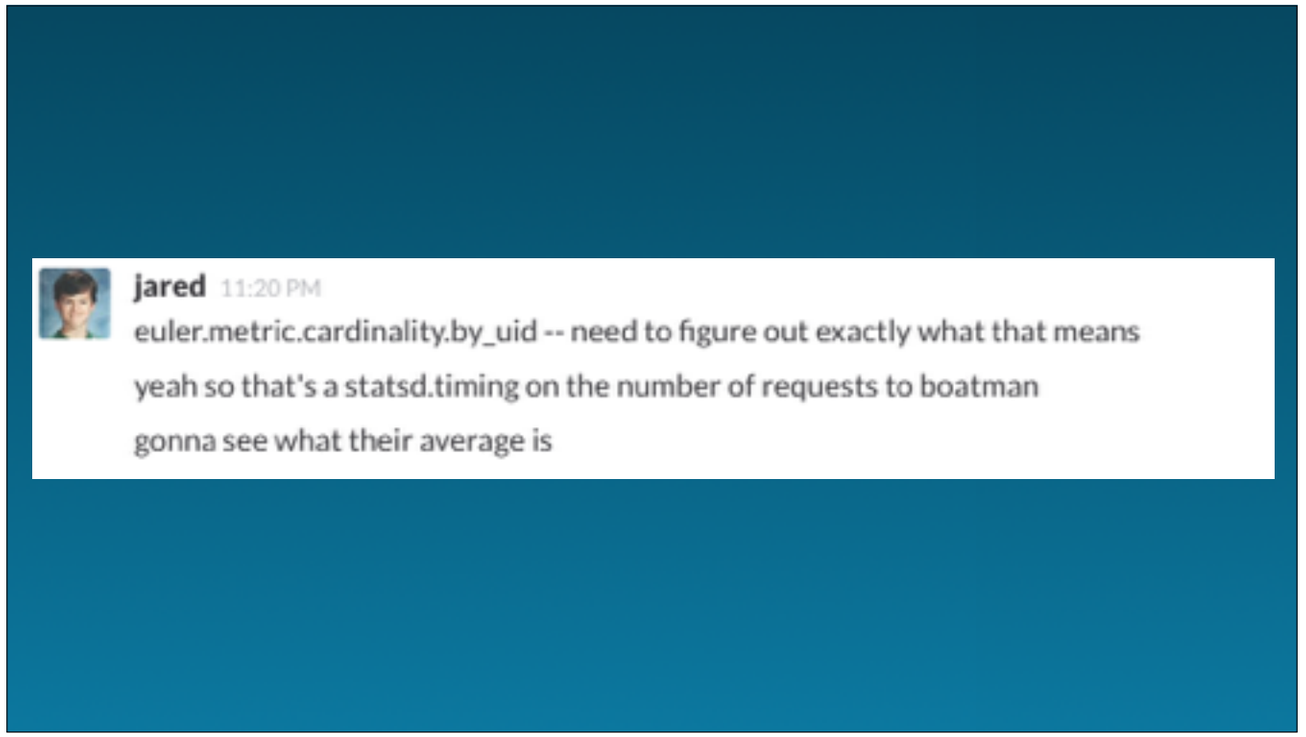
Right so you can imagine the front-end api is wired up to a queue, and this service isn’t popping stuff off the queue fast enough



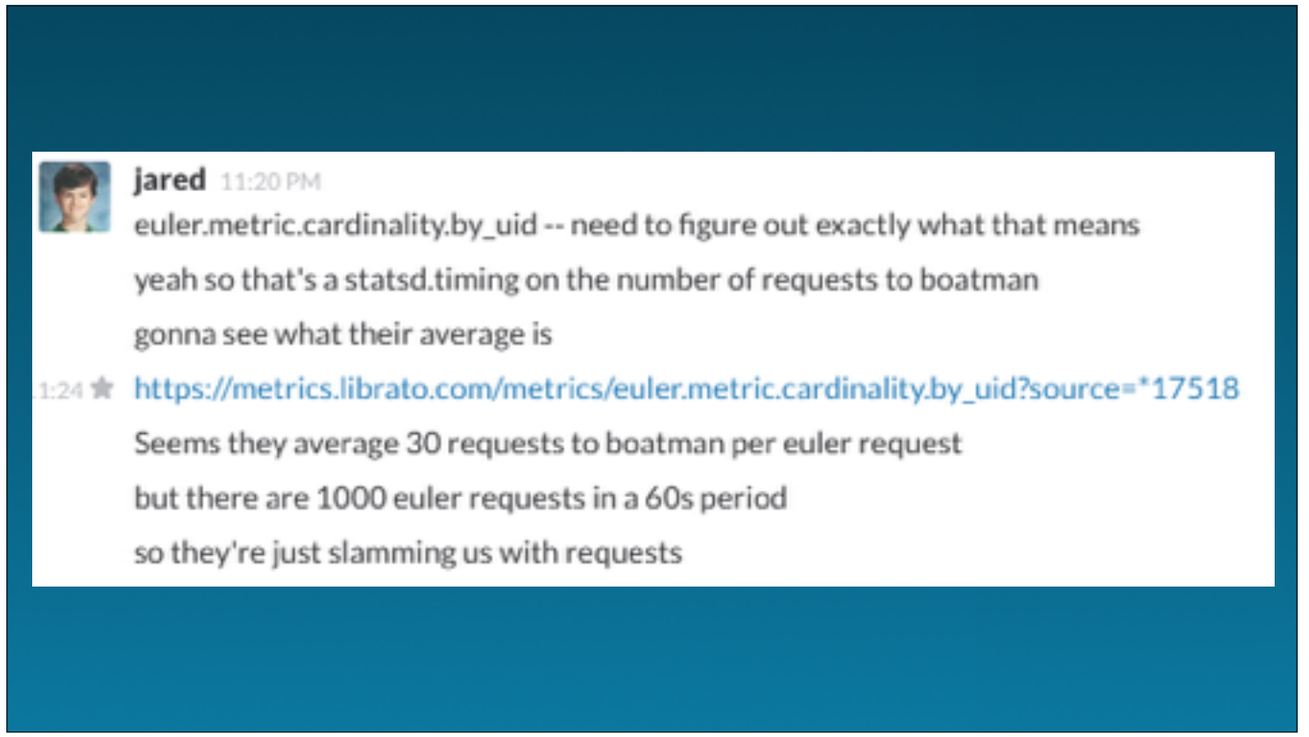
and that’s putting back-pressure on the queue, which is in-turn putting back-pressure on the API



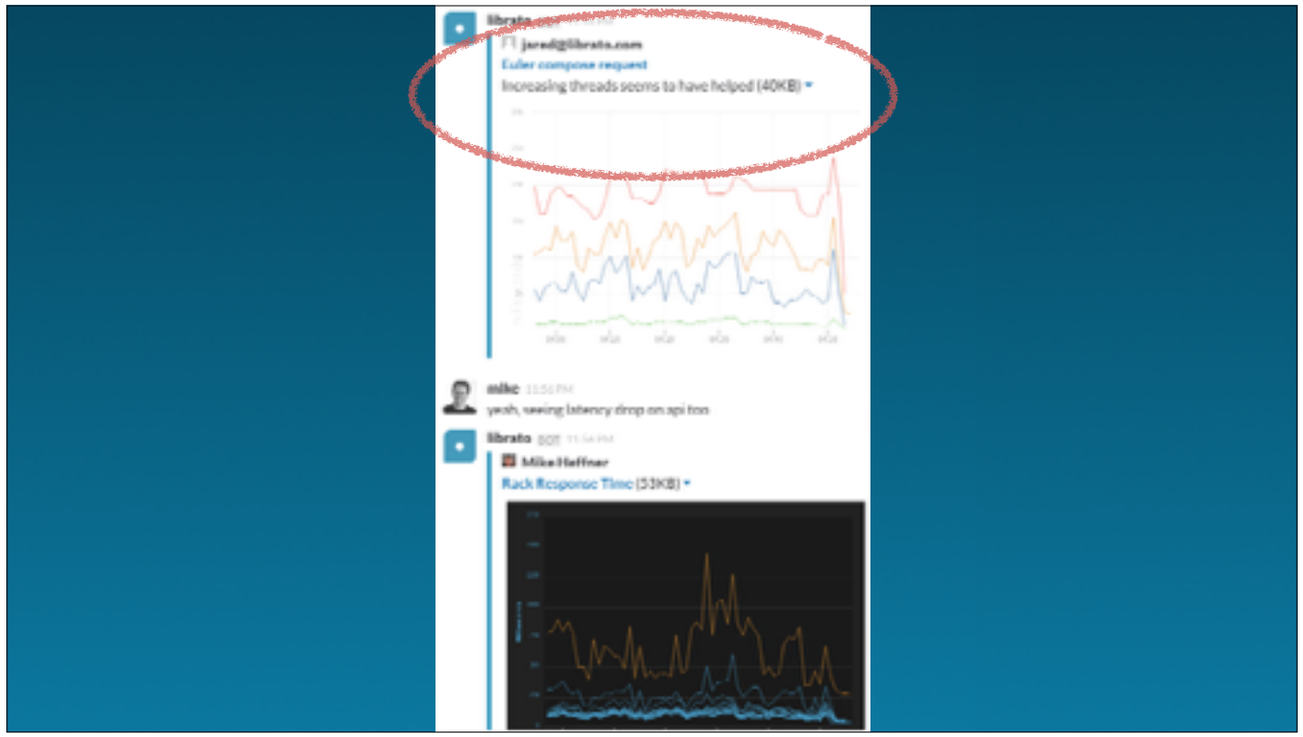
Jared suspects that this is because there are just too many user requests going on, so he looks at request cardinality and finds this, which yeah, looks… aberrant.



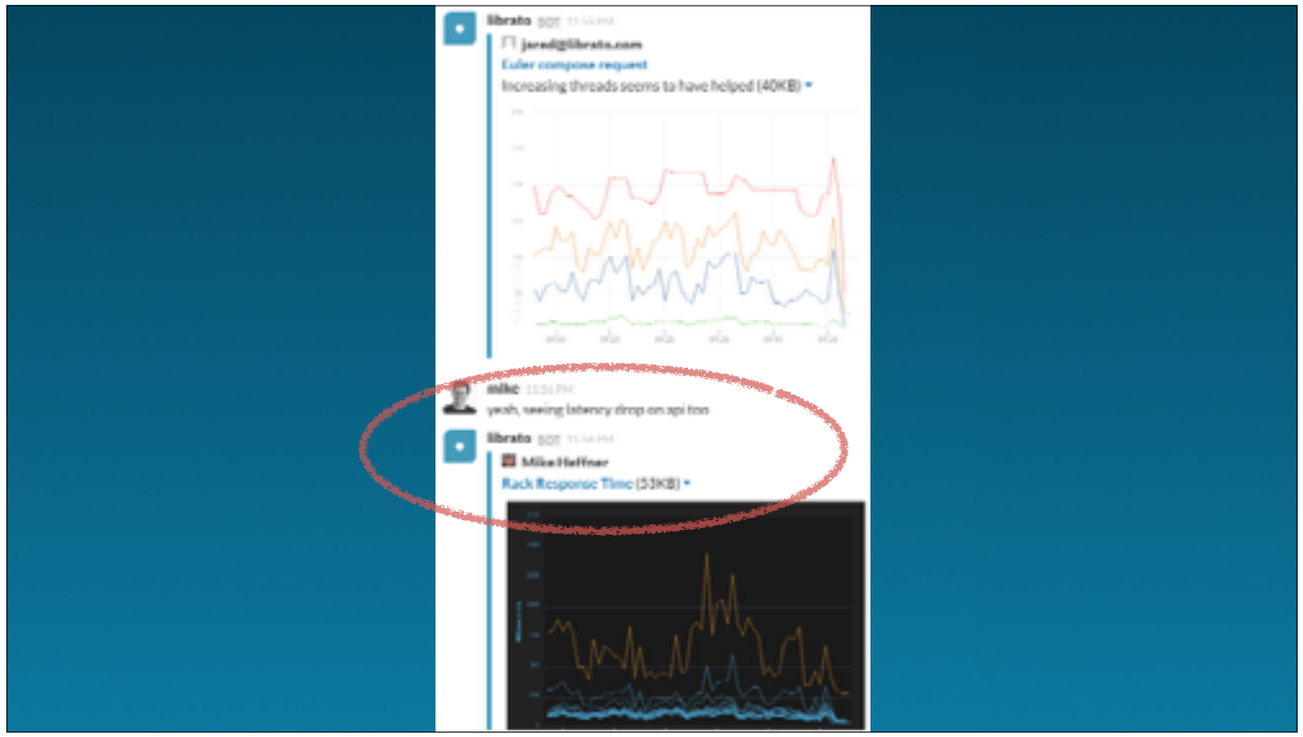
And this was a cool one for me to look at because Jared sees the cardinality data, but initially he’s not precisely sure how it’s being measured, so he actually goes to check the instrumentation code so he’s sure exactly what that line represents, how it’s measured and etc..



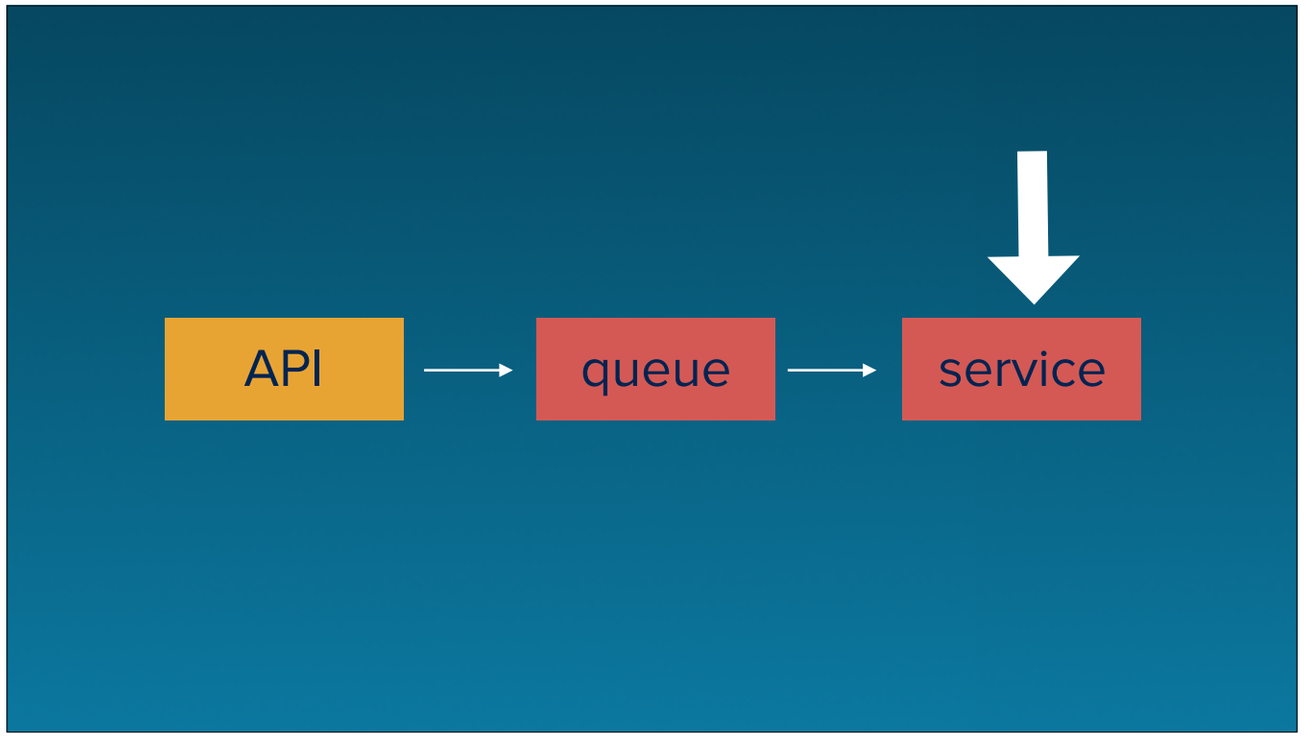
And then once he’s clear, he can follow up with correctly interpreting and visualizing that data.



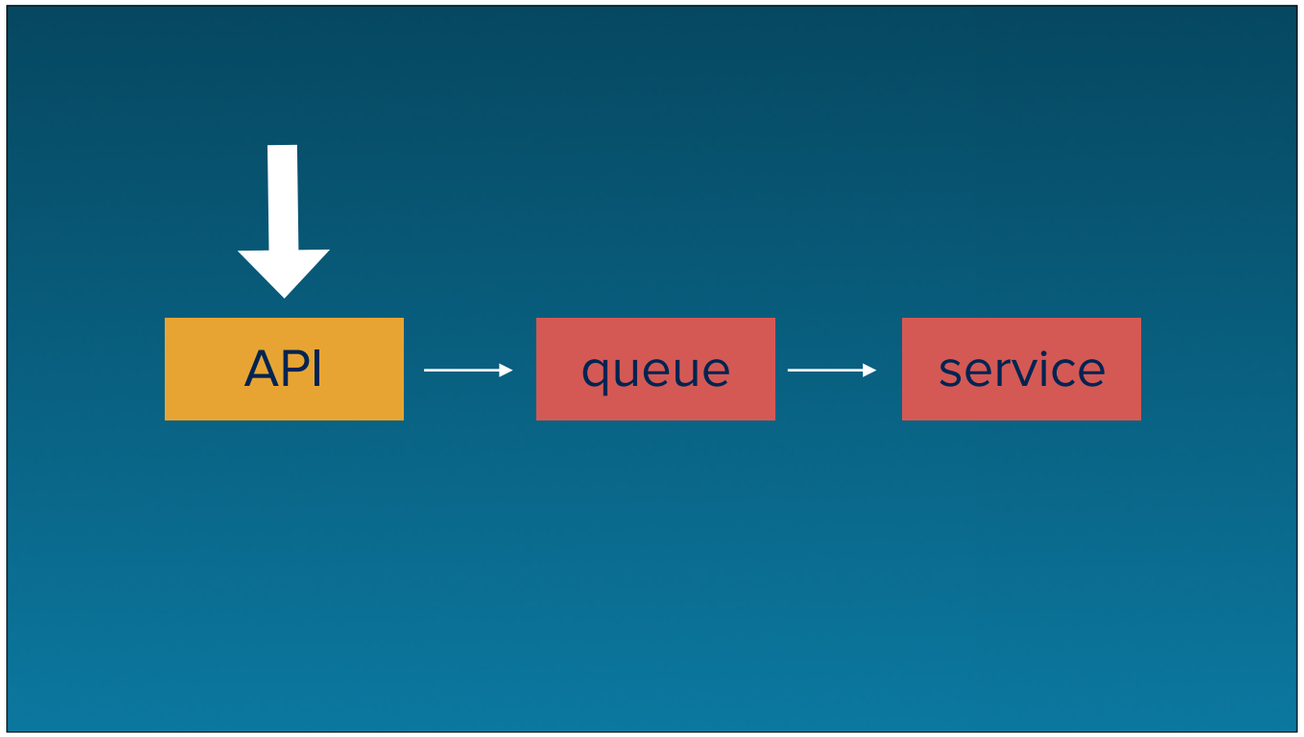
So anyway he pushes a change to bump the thread-count on this service, basically feeding it more workers, and here he’s using the telemetry data of the problem in progress, to observe the result of his change.



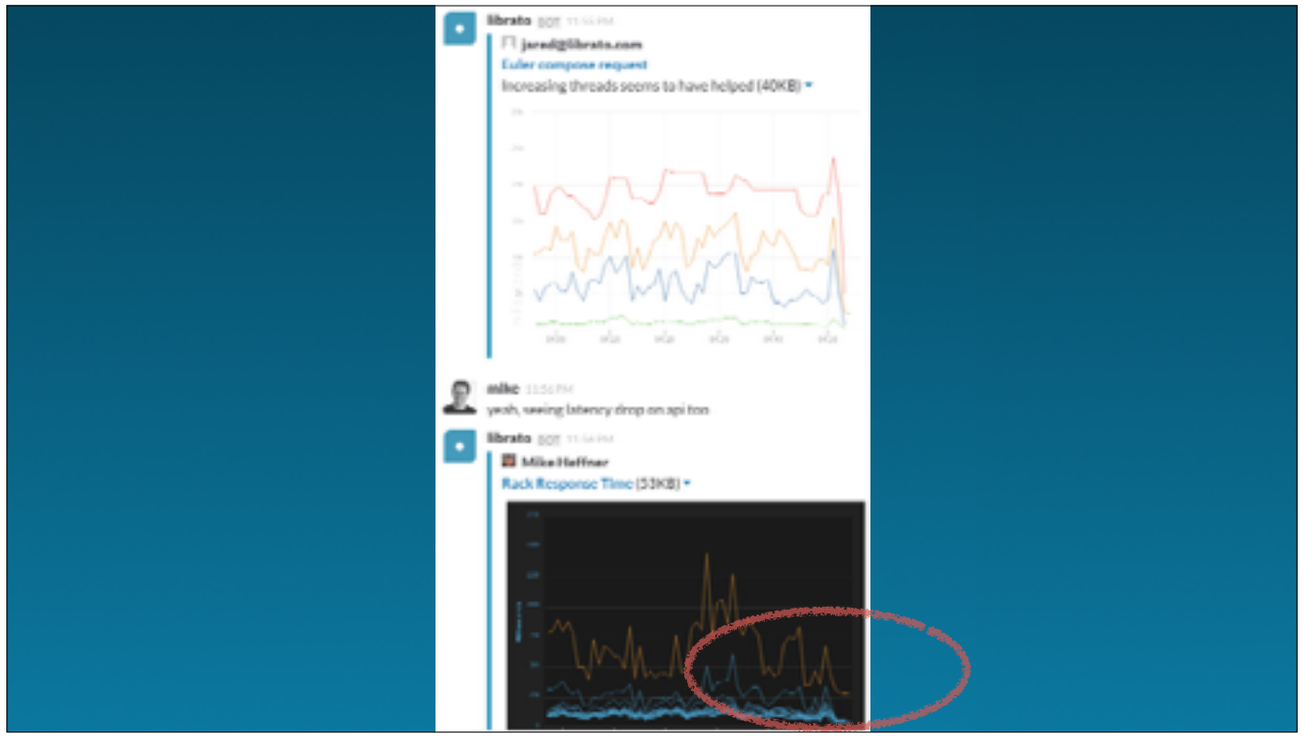
That looks promising, so Mike, moves forward toward the customer-facing side of the infrastructure,



remember, this was backing up into the API, so Jared was working here at the service, and looking at service-specific data



So Mike moves up here, to the API, basically asking, Ok if the data looks better at the service does it also look better up here at the API

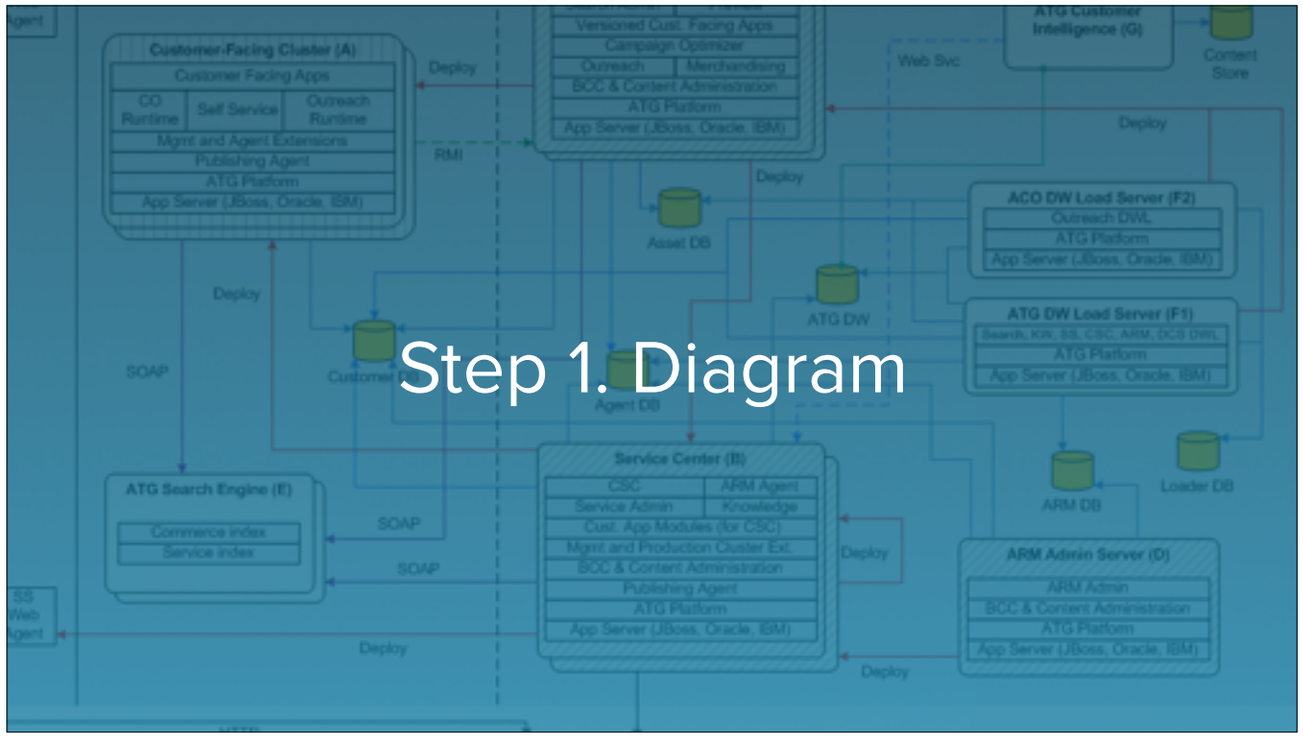


And yes you can see the API latency is slacking back off. So Jared's change has unblocked this service, which has in turn spead up the API response time as observed by our customers.

So in that little issue we used three different custom metrics, and it’s pretty normal for us to go through at least that many troubleshooting something at librato. We'll detect a problem in a front-facing custom metric, and dive in from there, tracking deeper and deeper into a service-specific custom metric. One leads to another which leads to another until we’re looking at the real problem. And just about every time, and I mean like high-ninetieth percentile, we have all the data we need to troubleshoot that issue. And we do this by-in-large by selecting individual metrics by hand as part of our engineering process

Homework

We don't measure everything, but I’m not going to pretend getting here is easy because obviously I personally failed to achieve it despite supposedly being a monitoring expert for a decade. So I want to talk you through a little homework assignment, and it’s a hard one, but if you do it, it’ll take you from being in charge of semicolons or wherever you are now, with whatever combination of tools that you have now, to a better place. Like maybe not Narnia, but I promise it’ll be better.

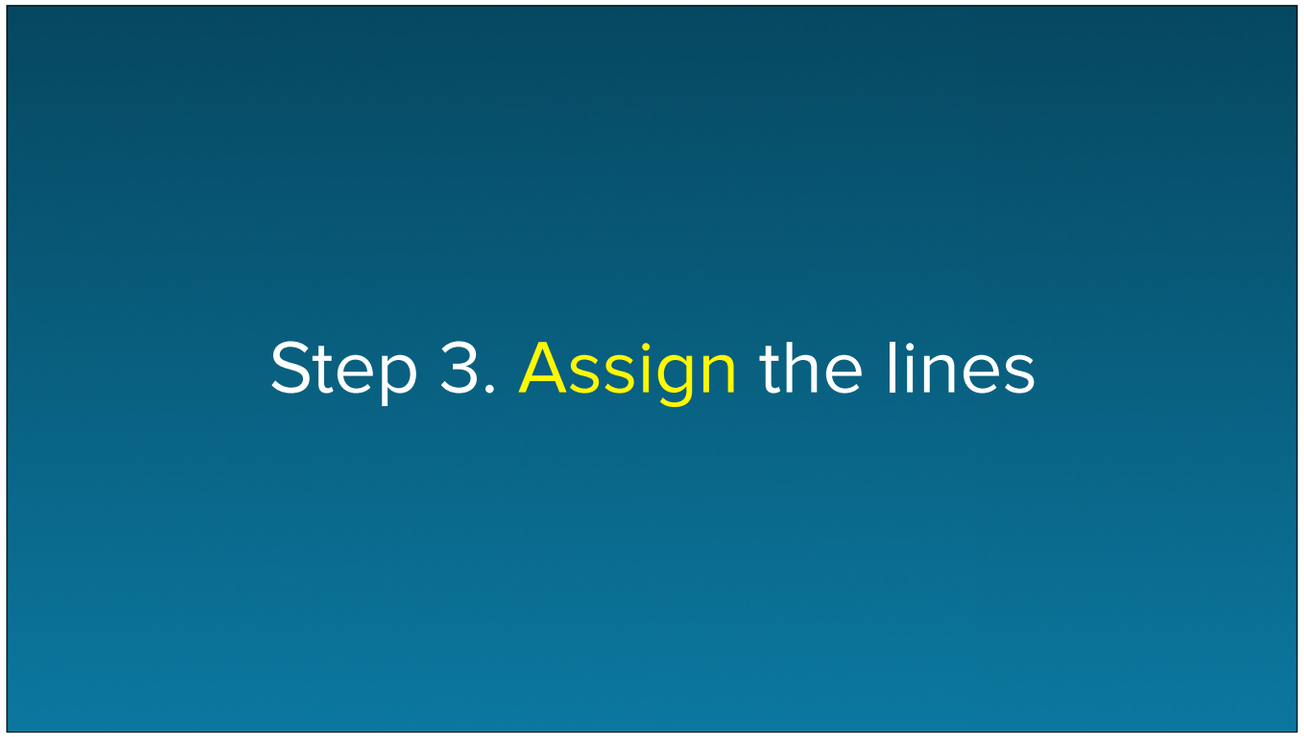


Step 1. Diagram

It begins with an architecture diagram So step 1. Acquire architecture diagram. It should have lots of little boxes and lines on it or possibly circles and lines on it.

Step 2. Label The Lines

step 2 is label the lines. Chances are you have a bunch of boxes with labels but, we’re more interested in the lines. We want to know what protocol each line represents. http, Sql, etc..



step 3. Put a name on each one of these lines. You need to know the person or people who you can talk to about each of these lines. And not like “the DB team” you want real people like Shannon or Joe. Like, Get out of your chair, walk over there, ask questions.

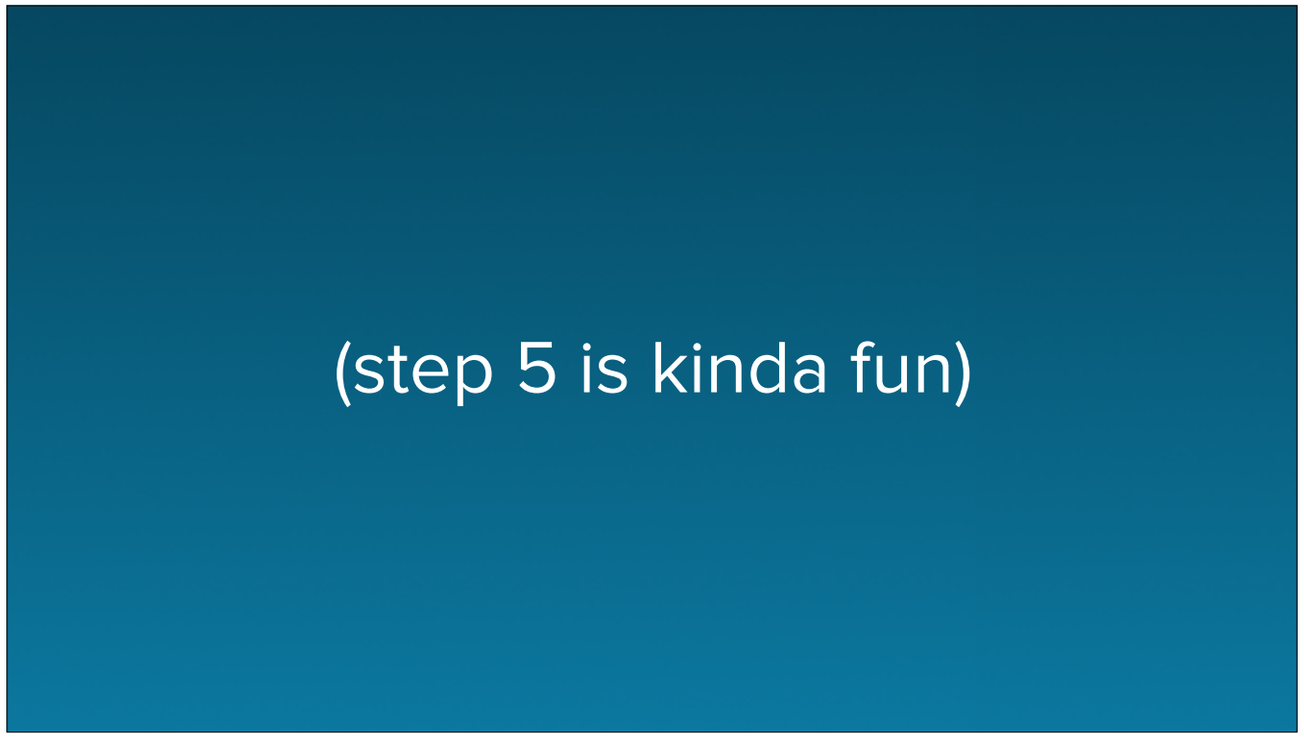
Step 4. Latency Data for Every Line

step 4. is Measure the latency of each one of these connections. So now that you know the services, the protocol they speak, and the people who know how they work, find an endpoint address, and use whatever you can get your hands on to get a response time. It doesn't matter how. Use windows. use node js, use freeking.. a commodore 64 if you have to. But you want a datastream that gives you the number of milliseconds that each one of these services takes to respond to a query. You want that data on the order of 60 seconds or less. Inter-service latency, data is SO useful. This is what lets you isolate just about any kind of problem to a particular layer of your infrastructure.

Step 4 is going to suck for you; it’s going to take you months. You'll probably have to throw up some new telemetry systems and data collectors. You're going to have to do a lot of people work. Have meetings, communicate. Stay on top of people. Follow up. On the technical side, you need to gather all of these numbers in the same place where you can graph it with a line graph. You're going to make sure that whatever that place is, anybody can send data to it, not just you. It can't be complicated, ideally it should support one of the defacto standard protocols to do this sort of thing like statsd or carbon.

Step 5. Talk about the patterns

step 5. is Watch the data. Don't set up any alerts, just check the numbers every day as you get more and more of them setup. You'll start to notice patterns, things that happen periodically, you won't be able to help it because of your monkeybrain. You'll also notice anomalies. Take note of both. Every single time you think you see a pattern, go to the person who owns that line and say "hey, check out this graph”.

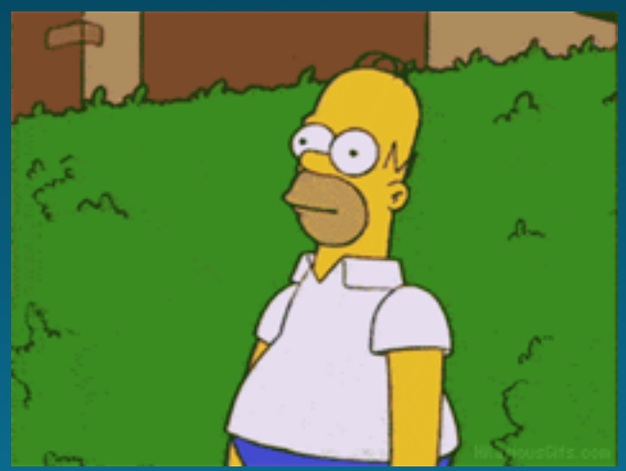


Keep in mind that step 5 sucks for Shannon and Joe. Very few people enjoy being confronted with shit they can’t explain. They’ll get fed up with you and roll their eyes and sigh when they see you coming their way, but that's fine because that means you're making them think about their service. And usually they'll say things like meh that's just backups, or whatever, but every now and then they'll say "hmm that's funny", and whenever you hear that proceed to step 6.

Step 6. Share the love

step 6. Is helping the service owner to add the instrumentation they need to explain the behavior they don't understand. Because some of what you’re seeing will be unexplainable given the data you have. That’s fine, so figure out what you need to add. Does that latency spike correlate with upstream requests? To answer that you’re going to have to quantify the requests. And that’s how real metrics happen. That’s how you choose for data over tools. This is also how you change the relationship, because Shannon and Joe get to learn about the monitoring system from you, and how to use it to figure out what’s going on, and you get to learn about how their service works. Then later, when stuff goes sideways, you'll be able to meet in the middle, and have a conversation about it.

Questions?



That’s about it for time I think, but I’d be happy to talk more pretty much anything in the hallway track! Thanks!