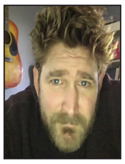
iVoyeur
Stacks and Piles

DAVE JOSEPHSEN



Dave Josephsen is a book author, code developer, and monitoring expert who works for Sparkpost. His continuing mission: to help engineers worldwide close the feedback loop.
dave-usenix@skeptech.org

I keep having this conversation with my coworkers. Honestly, it's probably to be expected given my penchant for harping on about monitoring tools. Also, I was admittedly quite spoiled at my last job, Librato—a place whose singular mission in life is operational visibility, where everyone has unfettered access to a functionally infinite, free, world-class, metrics platform—where things were, of course, different.

Anyway, the conversation I'm talking about usually starts off with me suggesting some tool that we could use to measure something. "Well how many foos per second are actually happening in real life?" I'll ask, expecting a number rather than a shrug in response. Alas, no one will know, so I'll suggest that we count them. "Do we have a graphite instance up anywhere?" I'll ask.

"No," they'll answer slightly annoyed, knowing full damn-well that I know full damn-well by now that there is no graphite instance, "we use Monitoring Tool X."

"Ah hah," I reply delighted, having successfully baited them into my personal little Platonic dialog. "But I'm not talking about monitoring, I'm talking about *measuring*."

Yes, delight. It delights me every single time, which, I recognize maybe is a little pathetic, but I'm already too old to care. In fact, one of the things I'm genuinely enjoying about the aging process is a certain sort of selfish introspection. It's great. You'll be walking down the street and suddenly realize that you keep on offering to meet people for a beer when you don't particularly like beer. And it just goes on like that, realization after realization that you've been engaging in all these behaviors that you kind of despise, and then, best of all, you just *stop doing those things*—like pretending to know what DevOps means, or living in Texas.

Anyway, most people don't really catch my meaning when I say I'm talking about *measuring things as an activity distinct from monitoring things*, so this portion of the conversation usually involves a lot of skeptical sideways glances and eye-rolling. And, honestly, I hear myself. I sound like a pompous windbag who swallowed a know-it-all jerk. The words emerging from my lips sound like something a televangelist might say if televangelists were really opinionated on the subject of IT monitoring tools. Like, these sentences could only emerge from the lips of someone who doesn't live *here*, in the bloody trenches with you and me. Someone who will soon jet back to the money-laden consulting partnership from which he oozed. I get that. I do. So the first thing I do is remind them what they have to go through to measure the number of foos traversing the wire with Monitoring Tool X.

First you need to know Monitoring Tool X itself: its YAML/XML/JSON/whatever configuration DSL along with its questionable world-view and unique collection of pseudo-random assumptions that I'm sure totally made sense at the time. Then, these days, there's usually a code promotion and review process, so you'll have to traverse those as well as possibly a change control process. Those things only apply if you're lucky enough to be allowed to actually *change* Monitoring Tool X. I don't have numbers, but I'm willing to bet that most engineers in most places aren't. Most engineers in 2017 still need to traverse a gatekeeper to affect Monitoring Tool X, which means filling out something akin to a trouble-ticket.

And so nobody measures.

Of course they don't. What carpenter would measure if she had to submit paperwork in XML before she could use the tape measure? Maybe someone would, but I would not hire that person and neither should you. I mean, at this point I've been dealing with monitoring system configuration syntax for over 20 years, and *I wouldn't* bother to measure if that alone was the bar to entry. I'd monitor, sure. But 10–15 minutes config time per new metric? I'd never *measure*.

But what's the big deal? I mean, ultimately, what do I lose? Obviously, we can *get by* without measuring. We can make things that work. Yesterday I walked in to my living room and brushed against a stack of recently purchased books in want of a shelf, but I did not knock them over. They teetered off balance, and, eventually, they might fall over as a result of their imbalance, but for the time being that stack remained a stack rather than a pile.

That stack is *working*. It's getting by. Exactly like so many other well-monitored tech-stacks in the interclouds. And when they fall over…when the stack becomes a pile, our monitoring tells us so and we intervene. Like a fire-alarm. That's how monitoring works. You don't want the fire-alarm going off when stuff isn't on fire, and so you restrict access to it, to make sure nobody messes it up.

That's not measuring. Measuring is what we do when we want to understand the things we build. How many queries is my service actually putting on the wire? How many threads does it spawn with real-life users? What's actually faster, the new parsing function or the old? Is round-robin actually round-robining (Hint: No)? Measuring invites us to answer these questions for ourselves. No paperwork. No fuss. Like a tape measure in our pocket, this is self-service. Nobody is worried about you breaking your tape measure.

When we measure, we can communicate actual, real-life systems behavior to one another, rather than hunches and estimates. Its output is truth. Not Warning, not Critical, just Truth. Measurement, therefore, gives us a common basis of understanding. It reaches across disciplines like application-development and ops (or SRE or whathaveyou) and provides a common comprehension of operational reality. Measurement gives us the ability to have objective conversations about the best way to fix things, and as your operational visibility improves, you begin to formulate a tangible sense of normality, and inversely, abnormality. You move from alerting on problems to detecting imbalance. You stop saying holy shit and start saying huh, that's weird, and seemingly overnight, you find yourself intervening before the stack falls over rather than scrambling to clean up piles.

Most importantly, measuring things changes *you*. It's one thing to read about the process versus thread model in Web servers, but it's quite another thing to see it for yourself. Measuring things, it turns out, removes the political subtext from our technology discussions. You no longer have to invest belief in the solutions for which you advocate. You are free to question and to formulate hypotheses and test them. It's habit forming, and it's a *really good* habit for an engineer.

From Logs to Sprites

A few days ago I participated in my first Hackathon at Sparkpost, and since I kept having this conversation, I thought I'd try to make something that celebrated the act of measuring as opposed to monitoring. Coincidentally, I've also been playing around lately with Phaser.io [1], a videogame development framework for HTML5-enabled browsers, so I thought I'd try to make a little traffic visualization toy.

DNS and SMTP are the lifeblood of Sparkpost, yet no second-scale metrics systems currently exist to visualize this traffic. Given this, I figured it would be impossible to render this traffic and not learn something in the process. I wanted to *show* everyone what our mail flow actually looked like, so I settled on SMTP and got coding.

Some 24ish hours later, Sparkviz was born, and I was super happy with how it came out. Here's a video of it in action [2].

On the far left, you see two Amazon ELBs: one balances inbound REST traffic from our customers and the other SMTP. This traffic is represented by green balls. The next tier inwards is our MTA tier. These servers relay mail outward to various proxies (the third tier), which in turn deliver to the Internet (represented as a large orange ball on the right). You'll notice the right-hand side of the screen is metered from 10 to 256. These obviously form a scale of first octets. Email successfully delivered appear as blue dots, which hit the far right-hand side of the screen at the point matching their destination IP's first octet.

The yellow balls represent transient bounces, and the red balls that impact the floor are permanent delivery errors. As the project took shape I noticed that heavy traffic often obscured patterns, so I used phaser's "enableDrag()" method on each of the sprites to make them draggable, as you can see in the video. When this wasn't quite enough I added a toggle to squelch out the errors entirely.

The project totaled 407 lines of code: 161 lines of JavaScript and 246 lines of Go. Unfortunately, I can't share it, but there's no reason it couldn't be open-sourced eventually.

It's implemented as a daemon designed to run on our internal log aggregation boxes. It listens on a UNIX domain socket for log-lines, which it parses and extracts into JSON blobs. You can see my highly technical architectural design document for the daemon in Figure 1.

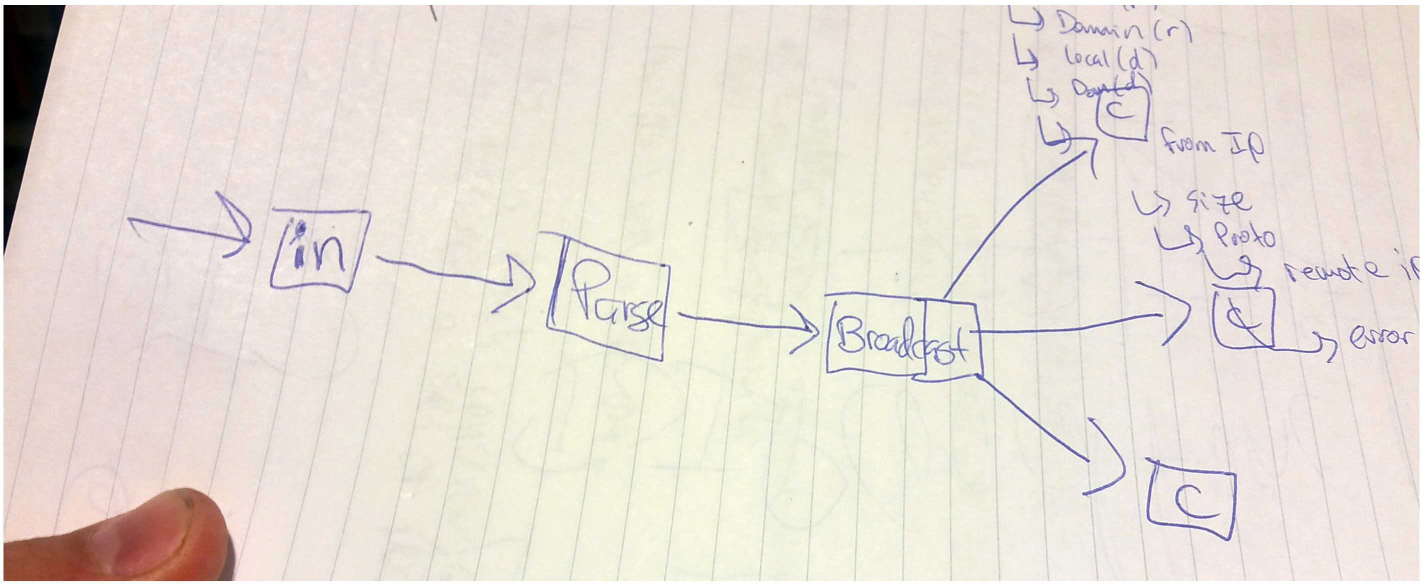


Figure 1: Highly technical architectural design document

The daemon also listens on port 8000 for HTTP clients, to whom it delivers the phaser-based JavaScript UI. The UI, running in the browser on the client, turns around and creates a WebSocket connection back to the server. The daemon keeps a globally scoped slice of these connected WebSockets and broadcasts each parsed log line to every connected client as a JSON blob (using a millisecond sleep function inside each client's broadcast go-routine to throttle the outbound traffic to 1000 blobs per client per second).

Differentiation of traffic type happens client-side, where the JavaScript UI uses a series of handler functions to parse out the event-type from each inbound JSON blob, pushing them on to another queue with the appropriate sprite value for phaser to render and tween. The tl;dr is that I created a firehose between the MTA logs and the end-user's browser. As always with hack-day projects, there's plenty of room for improvement, but as you can see, it gets the job done.

As I suspected, we all learned quite a bit from the exercise. It's kind of impossible for humans to avoid pattern-parsing data like this, and you don't need to look at it very long to recognize that we have a distribution imbalance in this environment. Certain MTAs clearly prefer certain proxies. Like the books in my living room, this stack works despite its imbalance. I, for one, am really looking forward to smugly pointing back to Sparkviz when I curmudgeonly lecture my contemporaries on the importance of operational telemetry, a process from which I'm sure I will extract far more than 407 lines of delight.

Take it easy.

References

[1] Phaser.io: http://phaser.io/.

[2] My traffic visualization tool in action: https://www.youtube.com/watch?v=htidm6DWq2s.