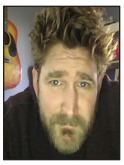
iVoyeur
Dogfood and the Art of Self-Awareness

DAVE JOSEPHSEN



Dave Josephsen is the sometime book-authoring Developer Evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

Yesterday, I ate lunch in a bar in Northwest Montana. I munched on their fish-and-chips plate (which was way better than it had any business being) and tried to ignore the *Bones* re-run playing in 4k clarity while the old-timers drank and argued behind me.

The argument concerned a certain very old copper mine with a long and storied history of screwing over everyone but their board of directors. I won't get into the politics of it with you, or bore you with my opinions, but suffice to say I could tell who was in the right, and I think you could too if you were there.

It wasn't so much the logic of the arguments, nor the passion with which they were delivered by either side. It was *the tone* used by those in the wrong—a certain manner of speaking that belies a particular mode of thought; I'm sure you would recognize it. I did. It was that same tone we used in the Marine Corps in those tiny moments of uncertainty that always accompanied our preparations to do a bad thing in the name of some supposed greater good. Even before that, though, I recognized it from the days of my youth, when I knew I'd done a bad thing but I was trying to convince myself, or someone else (or both), that I had a good reason.

It was that guilty-as-sin *yeah...but* tone. We can all recognize it in others as long as we've recognized it in ourselves; and we all have.

"Know thyself" was one of the Delphic maxims, did you know that? Literally carved in stone into the temple of Apollo at Delphi. It's one of our oldest and best thoughts; one of those things we've been thinking since we've been capable of thinking about good and bad.

Sorry if I'm being a bit of a downer, but I actually find that a really comforting thought, that our self-awareness carries with it a certain, well, inescapable self-awareness. All we have to do is pay attention to ourselves.

Speaking of self-awareness, here's an interesting but not often answered question from my current day job:

What's monitoring the monitoring system?

I know, that's the kind of question asked by people who want to sell you something, but it's also one of those questions that triggers a certain degree of guilt within those of us who don't have a good answer for it. That's why the pre-sales engineers love asking it. They intuit our guilt because they've recognized it in themselves.

But how important a question is it really? I suppose it depends. There's something of a continuum of monitoring aptitude. The shops at the baseline competency level don't really distinguish between monitoring and alerting. Monitoring *is* the system that sends alerts, so for them, a monitoring outage is an alerting outage. Those stakes aren't very high honestly. They may not be alerted to a problem, but once they do find out, they'll SSH into that system and poke around manually. That kind of sucks but it's not the end of the world.

Moving up the continuum, however, you begin to encounter shops that use monitoring as a means of understanding system behavior. By that I mean, when an operations person wants to know if the app slowness they're experiencing is isolated to them or a widespread issue, they turn to the monitoring system to find the 95th percentile latency on HTTP requests. Then maybe they'll break out that data by node to find a misbehaving instance, and tell the chatbot to destroy that instance and replace it with a new one. In those sorts of shops, a monitoring outage directly affects our ability to reason about and fix problems with our systems.

Losing visibility at that level in the continuum sucks even more, but at Librato, we're in an even worse pickle. The monitoring system is not only our primary means of understanding the behavior of our systems; it *is* our systems. We are a SaaS monitoring shop, so a monitoring outage here equates to a catastrophic business interruption. I know, weird, right? So in this issue I thought it might be fun to explore the question of what monitors the monitoring system at Librato.

The tl;dr is, of course: Librato, but the story of *how* is pretty interesting and, I think, worth telling. In fact, Librato is the result of a pivot [1] from a product called *Silverline*. Silverline was designed to dynamically adjust the performance characteristics of a machine image in order to save money on hosting costs (nerf-your-CPU-so-you-spend-less-as-a-service). The engineers who built Silverline obviously needed a scalable means of measuring granular system performance, and so, like so many shops before them, they built a custom metrics solution. However, unlike so many shops before them, they did a *really* good job of it, and Librato was born.

When metrics became the operational focus of the company, the engineers were already quite accustomed to having unfettered access to an essentially free, high-quality metrics and monitoring tool. For them, building a thing and measuring its performance were the same undertaking, so they naturally relied on Librato to build and maintain everything. Put more succinctly: they used Librato to monitor the operational characteristics of Librato, thereby becoming their own biggest customer.

I cannot recommend this strategy for your monitoring endeavors, but it worked out pretty well for us in practice. In many ways it was even quite beneficial. It certainly brought us closer to our customers, since literally every employee at Librato could provide customer support because everyone was using the tool every day. It also gave us a far more solid baseline understanding of the technical limitations of the system than most startups have, since we were the ones who were stressing it the hardest.

It wasn't long, however, before a few very large engineering shops signed up, and UID1 (as we affectionately refer to ourselves) was no longer even close to the most voluminous metrics source. And as anyone who maintains an API will attest, along with more and larger customers comes a certain amount of API abuse. So it was with us. It really is remarkable that after 20 years in the field, you can still be surprised by end-user behavior. Humbling, but remarkable.

Unexpected patterns of end-user behavior are an inevitability for which no Chaos-Monkey can prepare you. We saw customers doing things that we'd never imagined, because honestly, they're kind of unimaginable. Can you imagine a scenario where you'd need to insert an epoch timestamp into the *source name* of a metric when you were going to plot in a line graph where x is time/day anyway? I mean why would you ever need to create a new, unique metric and source object for every single measurement you take?

That's just one of the many, *many* real-life things that we've seen real-life customers do in real life.

With time-series [2] datastores you make certain assumptions about the cardinality of measurement *sources*. We can handle high-cardinality measurements as long as we can make assumptions like those. I won't bore you with the details, but the accidental generation of high-cardinality *sources* can really wreak havoc on an optimized datastore like ours. Really any sort of behavior that causes us to create rows on the order of millions of unique IDs (or anything else that isn't proper time-series data) per minute or second is basically guaranteed to trigger pager duty to wake me up in the pre-dawn.

When problems like that happen, we need to be able to get to our own metrics to diagnose the issue, and we can't do that if the system is being effectively DOS'd by another end user.

Enter Dogfood

Our solution to this problem is an environment we named *Dogfood* (in reference to that somewhat gross Microsoft colloquialism [3], eating your own dog food).

Dogfood is pretty much a mini-Librato—a miniature re-creation of our production environment just for our use. It resides on AWS in US-West, on the opposite coast of the US from our Production and Staging environments. It is fed data by way of our production stream-processing tier, which is a custom-built stream processing system we've talked publicly about in the past [4].

Well-implemented stream processing is a lovely thing, and at Librato we rely very heavily on the combination of SuperChief (our own beloved Storm replacement) and Kafka [5], which we use as a cache between our various stream-processing entities. These components make it possible for us to quickly persist raw-resolution measurements as they arrive to our API while simultaneously processing them for alerting and copying them over to the Dogfood environment.

The pattern is simple. Worker threads from one service take measurements off the wire and write them to Kafka queues (*topics* in Kafka parlance). Workers from other services read them out of queue and process them in parallel. A single measurement, for example, that hits our API is immediately copied to several places by our ingestion pipeline:

- Directly to a fast-path service designed to persist it in Cassandra
- To a Kafka topic read by the Alerting service (for alert detection)
- To a Kafka topic for the service that processes 60-second roll-ups
- To a Kafka topic read by Dogfood workers
- Kafka topics for other stuff I can't talk about yet

The Dogfood path is triggered for every measurement that's submitted by user: UID1. We implemented Dogfood duplication as a stream-processing job like this so that any metrics we create in our day-to-day work will automatically be picked up and sent to Dogfood. Monitoring systems succeed when they're easy to use, so I feel pretty strongly that this is a critical component to Dogfood's success. It just wouldn't work if, as an engineer, you had to remember to create every metric twice: once in production and once in Dogfood.

But What About Ingestion Pipeline Problems?

The downside of using production streaming infrastructure to tee off metrics to Dogfood is the possibility that we will have a critical blocking outage in the production stream processing tier that will affect Dogfood metrics. Problems like this are actually relatively rare given both the simplicity of Dogfood processing (it's just a single write operation) and the parallel nature of stream processing with Kafka. In fact the most wonderful thing about our stream processing is how well it isolates workloads from each other. Given separate Kafka topics and dedicated services behind each, it's very rare in practice for us to experience an issue that crosses multiple topics, much less unrelated ones. Those sorts of issues are pretty much always going to be upstream of us at AWS (where Dogfood won't help us anyway).

Another downside is the possibility of a problem in the API ingestion pipeline upstream of the stream-processing tier. If the metrics can't make it into the stream-processing tier, then they won't make it to Dogfood either. This is a more likely failure mode, and one that we've experienced in the past. In practice, however, because of the nature of our architecture, the absence of Dogfood metrics is a pretty damning indicator of an ingestion problem, so when this happens we already know exactly where to look.

Most of us prefer to use live data from production day-to-day because it's the same system our customers use, but if we ever experience a service degradation, we can switch to Dogfood seamlessly to diagnose the problem and work toward a fix. Dogfood might be the most elaborate answer ever to the question of *What's monitoring the monitoring system*, but then who can put a price on self-awareness?

Take it easy.

References

[1] Wikipedia, "Pivot," last modified on Sept. 13, 2016: https://en.wikipedia.org/wiki/Lean_startup#Pivot.

[2] Dave Josephsen, "Sensical Summarization of Time-Series" (blog entry), August 11, 2014: http://blog.librato.com/posts/time-series-data.

[3] Wikipedia, "Eating your own dog food," last modified on Sept. 3, 2016: https://en.wikipedia.org/wiki/Eating_your_own_dog_food.

[4] SuperChief: http://www.heavybit.com/library/blog/streamlining-distributed-stream-processing-with-superchief/.

[5] J. Shekhar and A. Khurana, "Streaming Systems and Architectures," *;login:*, vol. 41, no. 1 (Spring 2016): https://www.usenix.org/system/files/login/articles/login_spring16_03_shekhar.pdf.