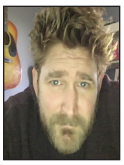
iVoyeur
Prometheus

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—a byproduct of my own incessant humble-bragging about living in Montana. I will be talking to someone I've just met, at a nerd meet-up say, or maybe a coffee shop or airplane, and the topic will just sort of come up. Arise. Spring forth into being.

It probably has something to do with my handshake. The way I grab people by their outstretched hand, not so much heartily greeting them as capturing them in place, and ensuring they cannot politely escape as I loudly exclaim something to the effect of:

"HI-I'M-DAVE-I-LIVE-IN-MONTANA!"

And then, as if by magic, the conversation careens away from whatever direction it probably should have been headed directly north into the tree-lined hinterlands.

Obviously, as the architect of this colloquial digression, I'm more or less rudely declaring my preference for hinterlands over the unknowable set of topics which could be derived from the unique soup of ingredients of your humanity combined with mine.

I mean, we were probably just going to talk about the weather anyway, but it does bother me. This notion that I have robbed us of the opportunity for spontaneous conversation in exchange for my personal, known baseline of locutionary enjoyment. Had I not grabbed the steering wheel and swerved, we might be talking about bread-making right now or tuning diesel engines. Maybe you're 3/4 of the way through the *Manga Guide to Linear Algebra* and are just dying to talk with someone about it. That would have been really fun, and I failed to allow that possibility to blossom between us. I didn't give you the credit you deserve.

I recognize that. I do. So although I brought us here, I won't insist on us staying. What often happens is that you will know of a person or place in Montana. You used to visit your grandma in Pony, or maybe your ex frequents the ice-climbing festivals in Hyalite Canyon. And so we are brought closer together by virtue of a shared experience with place.

Maybe the notion of wilderness itself resonates with you. Your heart pounds for the dry-hot, windy openness of the West: Arizona, New Mexico, Utah. Maybe you've wandered central Africa or sailed the coast of South America. You've never been to Montana per se, but there is a remote place threaded into the fabric of your soul. A hard-to-get-to place into which you could happily disappear and, just as happily, never return.

But just as often, you happen to be someone to whom my happy place parses as a hellish sort of prison. A person who finds it hard to imagine a fate worse than banishment to the icy, deserted foothills of the Absaroka-Beartooth Wilderness, where the elk outnumber people, there is no decent pizza, and the preponderance of what little conversation there is to be had revolves around, well, bread-making and tuning diesel engines.

Thus, a sort of conversational reset transpires. We can't, after all just throw our hands in the air and run away from each other. That'd be weird. So we do what all good engineers do: we turn it off and then turn it back on again. You say something like, "Wow, you really must like

getting away from it all," and I chuckle politely and hand over the wheel. First law of improv and all that; I'm sure we can find common ground in some other subject matter that is hopefully not the weather.

But that phrase *getting away from it all*; it'll still bug me weeks later. The unspoken thought that I am willingly abandoning the richness of a full place for one of emptiness. That I'm escaping or ejecting. Getting away from "it," whatever it is.

I guess maybe we have different *it*s. And I think I get yours, the cultural it, with the third-wave coffee and the music and the bright laughter, and liberality of ideas. The golden aura that glows around healthy, sophisticated, urban coexistence.

But you don't need to come to the hinterlands to experience my it. You can find it within yours. *Between it*. In empty offices after dark and in shops that haven't opened yet. In any kind of place where, if you were to stand in place and turn time like a knob, you would see that the place always exists, but the people who inhabit it exist only between the hours of 8 and 4 say, only as long as the band is playing or until the bell rings.

With apologies for polluting an otherwise objective and scientific journal with magical thinking: there's something that lives in the spaces we vacate. Not an emptiness, but a kind of presence or potential. A thing accentuated by our absence. A signal underlying the obvious stuff, there for anyone who cares enough to listen. A wilderness. You can feel it vibrating there if you hold still, in the in-between. That's my it.

Prometheus

I have mentioned Prometheus (http://prometheus.io) here and there in my recent articles, and I think the time has finally come to do a short series on it. There are myriad details of note. Its built-in time series database and cloud-native focus. Its reinvention (or maybe revitalization?) of the pull-model so abhorred by DevOps acolytes, and its query-centric operation and accompanying domain-specific language: PromQL.

But rather than charging head-first into any of those technical frontiers, I'd like to take a moment to make a point about what systems like Prometheus represent in the context of the evolution of systems monitoring. There is a continuum where, on one end, exist the monitoring systems of the "past." They were built, in a way, unconcerned with what they would eventually be tasked with monitoring. Which is fair, since the things they monitored were equally unconcerned with being monitored.

If we built airplanes like this, it would be as if we built one airplane with no instrument panel and then a second altogether separate airplane to fly beside it, taking note every so often of the first plane's speed and the temperature of its exhaust. Whatever could be seen from the outside. The first airplane flying happily, obliviously along until it didn't. And in the aftermath we'd ignore the wreckage, turning our attention instead to the plane that *didn't crash*. Scratching our heads and combining its samples with our intuition to guess at what might have happened to the plane that did.

On the other end of the continuum, every piece of software written worldwide *is a monitoring system* and, oh by the way, also possessed of some additional, arbitrary functionality. As if everything we make is instrument panel, upon which we sometimes bolt wheels or wings.

Both ends of this continuum are dystopian in their special way, but also instructive in that they hint at some middle ground in between. Imagine a place where applications run, interactions happen, and customers are serviced, and in between all of that, quite unbeknownst to everyone, a telemetry signal is emitted, underlying the obvious stuff. There for anyone who cares enough to listen.

We aren't building a monitoring system into everything we make, but we are acknowledging the importance of feedback, upfront in the development process, and providing an organizational answer to the question of where to send operational telemetry.

Prometheus server is probably the first centralized poller that assumes the presence of such a signal. It's a traditional poller in that it wakes up on a configurable interval, polls metrics, and stores them internally. Its contribution to the field of pollers is subtle but important: it *only* polls this specific kind of signal, emitted via HTTP on one or more TCP ports on every host.

If you've worked with other monitoring systems, this might seem like a limitation at first, but it's really quite liberating. There is no particular agent to install; no collector-side of its body to configure and reunite with its head. Instead, you are encouraged to participate directly in the data-model by taking whatever you want Prometheus to slurp up, packing it into a text format, and making it available via HTTP on a pre-arranged TCP port.

So we are left to create this signal. We can emit it from within—inside a running process or thread that is our application—or from the outside using a piece of software dedicated to collecting metrics. We can even post-process logs into the correct format. Obviously, we can also use a combination of techniques and, in practice, pretty much always do. Here's a simple example of its text format.

```
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 71
# HELP process_cpu_seconds_total Total user and system \
        CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 24738.72
# HELP process_max_fds Maximum number of open file \
        descriptors.
# TYPE process_max_fds gauge
process_max_fds 1024
```

These metrics were emitted from inside a Go program, using the Go exporter (https://github.com/prometheus/client_golang). If you're used to dealing with metrics systems, the concept of a named value like `process_max_fds 1024` is no doubt very familiar to you. Note also the metadata fields that provide a data-type and a human-readable description for each metric; although these appear to be comments, they are in fact required fields.

In Prometheus land, the software that emits metrics signals is always called an *exporter*. Getting started, you probably won't need to write one yourself because there are already off-the-shelf exporters for just about everything you can imagine, including a "node exporter," which works like a traditional monitoring agent, reading values out of /proc and /sys and exporting these as Prometheus signals for you.

Here's another, more complicated version of Prometheus's data format:

```
# HELP nginx_requests_total Total number of reqs by HTTP \
        status code.
# TYPE nginx_requests_total counter
promhttp_metric_handler_requests_total{code="200"}
1.654693e+06
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

This example adds "labels," enclosed in curly braces after the metric name. Arbitrary metrics dimensionality can be achieved with labels, but the Prometheus documentation (and anyone who has used it in anger) warns against their overuse (https://prometheus.io/docs/practices/naming/). In real life, metrics should be restricted to labels with fewer than 10 dimensions: e.g., recording requests by HTTP type is fine, but attempting to record requests by customer ID will quickly blow up in your face.

I can tell you firsthand, the act of creating and nurturing this signal proliferates quickly through an organization, with application engineers using libraries to export metrics directly from their services as well as more operations-oriented engineers writing shell-scripts or other little pieces of automation to add more host-oriented metrics. Once you start working with it and relying on it, it quickly becomes habit-forming. At Fastly we've written a discovery tool (https://vimeo.com/289893972) called *PromSD*, which makes it easy for Prometheus (and people like me) to discover and explore the metrics backchannel in the in-between space of our network.

The data model is probably my favorite thing about Prometheus, this lovely notion of a monitoring subtext quietly woven into the fabric of the "important stuff." A single text format we can all agree on, that multiple monitoring systems or even human beings can consume. It feels like our little secret, and I think it's an important step forward for the field of monitoring in general. Tune in next time when we'll explore Prometheus's local storage, and PromQL, the query language you can use to interrogate the storage layer and draw graphs.

Take it easy!