iVoyeur

7 Habits of Highly Effective Monitoring Systems

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

Having recently returned from Monitorama [1], I can attest that it is exactly what it sounds like: a collection of people so enamored of the technical discipline that has been my obsession for the better part of the last decade that they literally fly from the far corners of the world in order to shut themselves up in a single room and geek out about it for days. There are drunken diatribes about RabbitMQ in the context of metric transmission, hallway arguments about whether CPU percentage or load average is the superior metric of computational stress, and diabolical plots to compress time series data by converting it to frequency space. My point is, this conference could not be more custom-tailored to please me were we gathering in a fellowship quest to craft the ultimate bacon, lettuce, and tomato sandwich.

If there was a theme that permeated the event, I think it was to be found in the contrast between two very specific kinds of talk. The first type is the kind given by someone attempting to apply mathematical (usually statistical, but sometimes signal processing) techniques to detect aberrant behavior in time series data. These are always technical and, with a few notable exceptions [2], do not attempt to practically apply their findings via a tool the rest of us can experiment with. They customarily provide an overview of relevant mathematical techniques, usually beginning with simple thresholds, moving through standard deviation and various types of exponential moving averages like Holt-Winters, and winding up somewhere in the vicinity of forward decaying priority sampling. At this point, they usually throw up their hands, mutter something about domain-specific knowledge and monitoring data being a non-Gaussian distribution, and ask for questions.

The second type of talk is the kind given by an engineer who has implemented a monitoring system that seems to be working for them at the moment. It is often a tenuously wired together Frankenstein's monster that will almost certainly look different the next time we see it (which is fine if it's solving their problems). To be clear, I greatly enjoy both of these kinds of talks. If there were a cable channel that brought me only this content, I would never leave the house.

Automated fault detection is absolutely worth pursuing, I'm excited about it, and I have no doubt there will be breakthroughs as we get more eyes on it. Further, it's always fascinating to hear about the real-life trials and tribulations of my fellow plumbers who are holding things together in their respective corner of the Internet. Their every success is a ray of glorious hope that brightens my day.

Being repeatedly subjected to these two types of talks back to back, however, was, I have to admit, a little disheartening. The contrast between the cold mathematical certainty promised by the former type compared to the banal reality of the latter really got me thinking about the current state of monitoring as I've personally witnessed it. Aren't there real-life monitoring systems out there that are purposefully designed, elegantly engineered, and that meet 100% of the needs of every engineering team in their respective organization? Yes, as a matter of fact, I happen to know that there are well-engineered monitoring systems that world-class IT shops are happy with: Systems that sure would benefit from automatic fault

detection, but wouldn't be defined by it. Systems that are worlds away from the cobbled together collections of tools from the second kind of talk—the kind, I might add, that the preponderant quantity of attendees I spoke to are running. And yet, the monitoring systems I'm talking about are often composed of those same pieces, but somehow manage to become more than the sum of their parts.

I like to think I've done a good job of resisting the urge to pontificate about the state of monitoring in general in this column, focusing instead on interesting tools and techniques. It just seems presumptuous of me to tell you what to install and how, but Monitorama has left me with both a burning desire to spout off at the mouth about monitoring theory and the feeling that I may have been remiss in avoiding it in the past. So, I give you my take on the current state of how to monitor well, organized into seven habits that summarize what the good systems are doing right today.

Habit 1: It's About the Data

Were I in a darker mood, I might have titled this "Stop Looking for an Ubertool." Awesome monitoring systems value data over tools—they understand that a monitoring tool is merely a means to obtain data. They treat metrics and telemetry data as first-class citizens and rarely leave it to rot within the tool that collected it. Rather, they send the data "up" to be processed, stored, and analyzed together with all of the other data collected by all the other tools, on all the other systems, organization-wide.

When you make the data a first-class citizen, you wind up with data-centric tools that enable you to correlate measurements taken from any layer of the stack. You can, for example, quantify the effect of JVM garbage collection on service latency, or if the number of calls to the foo() function in your application across three different nodes correlates to the odd behavior in the byte counter that resides on the switch they are all connected to. You know you are doing it right when you can "tee" off a subset of your monitoring data at will and send it as input to any new tool you might decide to use in whatever format that tool expects.

Now that I've made a big deal about it not being about tools, let's talk about the kinds of tools that let data thrive, beginning with an example of what not to do. I'll go ahead and pick on Nagios for this, since that's so in-vogue these days. Nagios was designed for a very specific job, namely, to collect availability data on services and hosts on the order of minutes (usually about every five minutes).

This is useful data to collect, and Nagios is, in my opinion, the best tool for accomplishing this task. It also makes some annoying assumptions about how you want to process the data it collects, and those assumptions make it more difficult than it should be to get data out of Nagios and into other tools. This is evidenced by the plethora of single-purpose tools that have sprung into being for no other purpose than to take data from Nagios and place it in X, where X is some other monitoring tool from which it is usually even more difficult to extract the monitoring data.

And so it is that we devolve into this anti-pattern of implementing the tool we think we need, and then more tools to connect our tool to yet other tools in an attempt to make up for some deficiency in the one we thought we wanted. The complexity of our monitoring efforts grows quadratically as our chosen tool bogs down with every new tool we bolt onto it. God help us if we ever want to connect a tool to the tool that's connected to the original tool, because our data just gets more and more specific, ever-increasingly locked-in to the toolchain we've painted ourselves into.

If, however, we recognize that Nagios is merely one of many data collectors and place a transmission layer above Nagios that is designed to accept metrics data from any sort of data collector so that it can be processed and persisted in a common data format, our tools no longer depend on each other, and we have a single source of telemetry data that we can wire to any tools that make sense. Obviously, I think this is a fantastic idea, and I even began to implement it myself [3] before Riemann [4] and Heka [5] did a much better job of it.

Habit 2: Use Monitoring for Feedback

Who is choosing your metrics? Are you using a turnkey agent that collects umpteen hundred metrics from every node that you install it on? How many of those metrics do you track? How many do you alert on? Great monitoring systems are driven by purpose. They are designed to provide operational feedback about production systems to people who understand how those systems work—people who have chosen what to monitor about those systems based on that knowledge.

Monitoring isn't a "thing"; it does not stand on its own. It is not a backup system or a disaster recovery plan, or any other sort of expensive and annoying burden heaped on Ops to satisfy the checklist requirements of a regulatory body or an arbitrary quarterly goal. It is not a ritual that grown-ups tell us to follow—like keeping our hands and arms inside the vehicle at all times—a habit we all must perform to stave off some nameless danger that no one can quite articulate.

Monitoring is an engineering tool. It exists to provide closed-loop feedback from engineering systems. It is the pressure meter on your propane tank. Through monitoring, we gain visibility into places we cannot go, and we prevent explosions from happening in those places. The engineers in your organization should understand the metrics you monitor, because each should have been configured by an engineer to answer a specific question or provide a concrete insight about the operational characteristics of your service.

Habit 3: Alert on What You Draw

When an engineer in your organization receives an alert from a monitoring system, and moves to examine a graph of monitoring data to analyze and isolate the problem, it's critically important that the same data was used to generate both the alert and the graph. If, for example, you're using Nagios to check and alert, and Ganglia to draw the graphs, you're raising the likelihood of uncertainty, stress, and human error during the critically important time of incident response.

One monitoring system or the other could be generating false positives or negatives; they could each be monitoring subtly different things under the guise of the same name, or they could be measuring the same thing in subtly different ways. It actually doesn't matter, because there is likely no way to objectively tell which system is correct without a substantial effort, and even if you do figure out which is lying, it's unlikely you will be able to take a meaningful corrective action to synchronize the behavior of the systems.

Ultimately, what you've done is shifted the problem from "improve an unreliable monitoring system" to "make two unreliable monitoring systems agree with each other in every case." The inevitable result is simply that your engineers will begin to ignore both monitoring systems because neither can be trusted.

Great monitoring systems require a single source of truth. In the current example, the most expedient way to achieve this is to configure Nagios to monitor thresholds in Ganglia's data [6] (because Ganglia has the best resolution). The concept of a single source of truth is a fundamental requirement to good systems monitoring. It's also another great argument in favor of focusing on data rather than tools.

Habit 4: Standardize Processing, but Emancipate Collection

I've run into business consultants who were convinced that the proper way to implement monitoring solutions was to first create a plan that lists every possible service that you could ever want to monitor and then choose a tool that meets your data collection list. In my experience, great monitoring systems do the opposite. They plan and build a substrate—a common, organization-wide service for processing telemetry data from monitoring systems—like the ones I described above in Habit #1. Then they enable and encourage every engineer, regardless of team affiliation or title, to send monitoring data to it by whatever means necessary.

Awesome monitoring systems standardize the metrics processing, storage, analysis, and visualization tools, but they declare open season on data collectors. One shop whose engineers I've spoken with (apologies, I've forgotten which) has the motto "new metrics in minutes." Every engineer should be free to implement whatever means she deems appropriate to monitor the services she's responsible for. Monitoring new stuff should be hassle-free.

Habit 5: Let the Consumers Curate

Another popular notion about monitoring systems in the corporate world is that they should provide a "single pane of glass," by which I assume they mean the monitoring system should have a single, primary dashboard that shows a high-level overview of the entire system state.

That's great and I'm not necessarily arguing against it, but the best monitoring tools I've seen focus instead on enabling engineers to create and manage their own dashboards, thresholds, and notifications. If you're doing it right, you should have a dashboard for every service that your team supports or contributes to, curated by your team members. Effective monitoring systems don't just allow non-ops engineers to interact with the system, they demand it.

Great monitoring systems are timely, open, and precise. They represent a single source of truth that is so compelling and easy to interact with that the engineers naturally rely on them to understand what's going on in production. When they want to track how long a function takes to execute in production, they should naturally choose to instrument their code and observe feedback using the monitoring system. When they have an outage, their first thought should be to turn to the dashboard for that service before they attempt to ssh to one of the hosts they suspect is involved.

A monitoring system that requires coercion for adoption isn't solving the right problems. So, if your engineers are avoiding the monitoring system, or ignoring it, or rolling their own tools to work around it, then you have an impedance problem, and you should ask yourself why they prefer the tools they do over yours, and focus in on making it easier for the consumers of the system to use it to solve their problems.

Habit 6: Evolve by Tiny Iterations

Healthy monitoring systems don't need a semi-monthly maintenance procedure. They stay relevant because they're constantly being iterated by the engineers who rely on them to solve everyday problems. New metrics are added by engineers who are instrumenting a new service or trying to understand the behavior of some misanthropic piece of infrastructure or code. Measurements are removed when they're no longer needed by the team that put them there—because they're superfluous and cluttering up the dashboards.

By focusing on the data, relying on the accuracy of the results, and enabling everyone to iteratively fix the pieces they rely on, your monitoring system will evolve into exactly what your organization needs it to be, rather than a complicated ball of cherished tools tenuously strung together that everyone ignores except the dude holding the string.

Habit 7: Instrumentation != Debugging

Monitoring is unit testing for operations. For all distributed applications—and, I'd argue, for a great deal of traditional services—it is the best if not the only way to verify that your design and engineering assumptions bear out in production.

Further, instrumentation is the only way to gather in-process metrics that directly correspond to the well-being and performance of your production applications.

Therefore, instrumentation is code. It is a legitimate part of your application—not extraneous debugging rubbish that can be slovenly implemented with the implicit assumption that it will be removed later. Your engineers should have libraries at their disposal that enable them to thoughtfully and easily instrument their application in a way that is commonly understood and repeatable. Libraries like Coda Hale Metrics [7] are a fantastic choice if you don't want to roll your own. In the same way your feature isn't complete until you provide a test for it, your application is not complete until it is instrumented so that its inner workings can be verified by the monitoring data stream.

As always, I hope you found something helpful in this diatribe. As the DevOps revolution continues to utterly confound and mystify the IT managementosphere, I think we have a golden opportunity to reinvent monitoring. My hope is that we can expand it from a thing that operations does because: computers, and replace it with a commons—supported by Ops—that welcomes measurements from every type of engineer and encourages them to define their own interactions, no matter how convoluted their title. To the extent we achieve this, I believe we will improve the transparency of both our services and infrastructure, increase our understanding of the systems we support, and carry with us quieter pagers.

Good luck!

References

[1] The Monitorama conference: http://monitorama.com.

[2] Abe Stanway, Jon Cowie: "Bring the Noise," Velocity Santa Clara 2013: https://www.youtube.com/watch?v=3nF426i0cBc.

[3] Hearsay: https://github.com/djosephsen/Hearsay.

[4] Riemann: http://riemann.io/.

[5] Mozilla, Introducing Heka: http://blog.mozilla.org/services/2013/04/30/introducing-heka/.

[6] Monitoring Ganglia data from Nagios: https://github.com/ganglia/monitor-core/wiki/Integrating-Ganglia-with-Nagios.

[7] Coda Hale Metrics: http://metrics.codahale.com/.