Monitoring in a DevOps World

**PERFECT SHOULD
NEVER BE THE
ENEMY OF BETTER.**

THEO SCHLOSSNAGLE, CIRCONUS

The title of this article might suggest that it is about how you are supposed to be monitoring systems in an organization that is making or has already made the transformation into DevOps. Actually, this is an article to make you think about how computing has changed and how your concept of monitoring perhaps needs recentring before it even applies to the brave new world of DevOps.

The more punishing truth is that this is not just a brave, but also a fast, new world. One of the primary drivers for adopting DevOps is speed—particularly, the reduction of risk at speed. An organization has to make many changes to accommodate this. The DevOps community often talks about automation and culture. This makes a lot of sense, as automation is where speed comes from and every problem can always be rephrased as a people (or communications) problem; automation and culture are key.

That said, the ground has shifted under the monitoring industry. This seismic change has caused existing tools to change and new tools to emerge in the monitoring space, but that alone will not deliver us into the low-risk world

Monitoring, at its heart, is about observing and determining the behavior of systems, often with an ulterior motive of determining “correctness.”

of DevOps—not without new and updated thinking. Why? Change.

Monitoring, at its heart, is about observing and determining the behavior of systems, often with an ulterior motive of determining “correctness.” Its purpose is to answer the ever-present question. Are my systems doing what they are supposed to? It’s also worth mentioning that *systems* is a very generic term, and in healthy organizations, systems are seen in a far wider scope than just computers and computing services; they include sales, marketing and finance, alongside other “business units,” so the business is seen as the complex interdependent system it truly is. That is, good monitoring can help people take a truly *systems view* of not only systems, but also organizations.

Long dead are the systems that age like fine wine. Today’s systems are born in an agile world and remain fluid to accommodate changes in both the supplier and the consumer landscape. A legitimate response to “adapt or die” is “I’ll do DevOps!” This highly dynamic system stands to challenge traditional monitoring paradigms.

THE OLD WORLD

In a world with “slow” release cycles (often between 6 and 18 months), making software operational was an interesting challenge. The system deployed at the beginning of a release looked a lot like the same system several months later. It’s not that it was stuck in time, but more that it was branched into a maintenance-only mindset. With maintenance comes bug fixes and even performance enhancements, but not new features, new

systems components, removal of old systems components, and new features or functions that would fundamentally change the stress on the architecture. Simply put, it isn't very fluid.

For monitoring, this lack of fluidity is fantastic. If the system today is the system tomorrow and the exercise that system does today is largely the same tomorrow, then developing a set of expectations around how the system should behave becomes quite natural. From a more pragmatic point of view, the baselines developed by observing the behavior of the system's components will very likely live long, useful lives.

This article is not going to dive into the risks involved with releasing a dramatic set of code changes infrequently, as there are countless stories (anecdotal and otherwise) that state their magnitude and probabilistic certainty. Suffice it to say: there be dragons on that path. This is one of the many reasons that agile, Kanban, and other more responsive work processes have been so widely adopted. DevOps is the organizational structure that makes the transformation possible.

THE NEW WORLD

So, we're all on board with rapid and fluid business and development processes, and we have continuous "everything" to let us manage risk. The world is wonderful, right? Well, "continuous monitoring" (in this new sense of continuous) doesn't exist, and, besides, the name would be pretty dumb; shouldn't all monitoring have always been continuous?

The big problem here is that the fundamental principles

that power monitoring, the very methods that judge if your machine is behaving itself, require an understanding of what good behavior looks like. Whether you are building statistical baselines, using formal models, or just winging it, in order to understand if systems are misbehaving, you need to know what it looks like when they are behaving.

In this new world, you not only have fluid development processes that can introduce change on a continual basis, but you also have adopted a microservices-systems architecture pattern. Microservices simply dictate that the solution to a specific technical problem should be isolated to a network-accessible service with clearly defined interfaces such that the service has freedom. Many developers like this model, as they are given more autonomy in the design of the service, extending to choice of language, database technology, etc. This freedom is very powerful, but its true value lies in decoupling release schedules and maintenance, and allowing for independent higher-level decisions around security, resiliency, and compliance.

This might seem like an odd tangent, but the conflation of these two changes results in something quite unexpected for the world of monitoring: the system of today neither looks like nor should behave like the system of tomorrow.

AN ASIDE ON ML AND AI

Many monitoring companies have been struggling to keep up with the nature of ephemeral architecture. Nodes come and go, and architectures dynamically resize from one minute to the next in an attempt to meet growing

A hot topic in monitoring is how to apply ML (machine learning) and AI (artificial intelligence) to the problems at hand.

and shrinking demand. As nodes spin up and subsequently disappear, monitoring solutions must accommodate. While some old monitoring systems struggle with this concept, most modern systems take this type of dynamic systems sizing in stride.

The second, and largely unmet, challenge is the dynamic nature not of an architecture's size but, rather, of its design. With microservices-based architectures and multiple agile teams continually releasing software and services, the design of modern architecture is constantly in transition.

A hot topic in monitoring is how to apply ML (machine learning) and AI (artificial intelligence) to the problems at hand, but the current approaches seem to be attempting to solve yesterday's problems and not tomorrow's. AI and ML provide an exceptionally rich new set of techniques to solve problems and will undoubtedly prove instrumental in the monitoring world, but the problems they must tackle are not that of modeling an architecture and learning to guide its operations. The architecture it learns today will have changed by tomorrow, and any guidance will be antiquated. Instead, to make a significant impact, AI and ML approaches need to take a step back and help guide processes and design.

CHARACTERISTICS OF SUCCESSFUL MONITORING

It would be cruel to cast a gloomy shadow on the state of monitoring without providing some tactical advice. Luckily, many people are monitoring their systems exceptionally well. Here is what they have in common:

What is more important than how

The first thing to remember is that all the tools in the world will not help you detect bad behavior if you are looking at the wrong things. Be wary of tools that come with prescribed monitoring for complex assembled systems; rarely are systems in the tech industry assembled and used in the same way at two different organizations. The likely scenario is that the monitoring will seem useless, but in some cases it may provide a false confidence that the systems are functioning well.

When it comes to monitoring the “right thing,” always look at your business from the top down. The technical systems the organization operates are only provisioned and operated to meet some stated business goal. Start by monitoring whether that goal is being met. A tongue-in-cheek example: always monitor the payroll system, because if you aren’t getting paid, what’s the point?

Mathematics: It’s necessary

Second, embrace mathematics. In modern times, functionality is table stakes; it isn’t enough that the system is working, it must be working well. It is a rare day when you have an important monitor that consumes a boolean value “good” or “bad.” Most often, systems are being monitored around delivered performance, so the consumed values (or indicators) are numbers and often latencies (a time representing how long a specific operation took). You’re dealing with numbers now, so math is required, like it or not. Basic statistics are a fundamental requirement for both asking and interpreting the answers to questions about the behavior of systems.

As systems grow and the focus turns more to their behavior, data volumes rise. In seven years, Circonus has experienced an increase in data volume of almost seven orders of magnitude. Some people still monitor systems by taking a measurement from them every minute or so, but more and more people are actually observing what their systems are doing. This results in millions or tens of millions of measurements per second on standard servers. People tend not to solve hard problems unless the answers are valuable. Handling 10 million measurements per second from a single server when you might have thousands of servers might sound like overkill, but people are doing it because the technology exists that makes the cost of finding the answers less than the value of those answers. People do it because they are able to run better, faster systems and beat the competition. To handle data at that volume, you must also use a capable set of tools. To form intelligent questions around data at this volume, you must embrace mathematics.

As you might imagine, without a set of tools to help you perform fast, accurate, and appropriate mathematical analysis against your observed data, you will be at a considerable disadvantage; luckily, there are myriad choices from Python and R to tools that will help you find more comprehensive solutions from modern monitoring vendors.

Data retention

A third important characteristic of successful monitoring systems is data retention. Monitoring data has often been considered low value and high cost and is often expunged

The final piece of advice for a successful monitoring system is to be specific about what success looks like.

with impudence. Times have changed, and, as with all things computing, the cost of storing data has fallen dramatically. More importantly, DevOps have changed the value of long-term retention of this data. DevOps is a culture of learning. When things go wrong, and they always do, it is critical to have a robust process for interrogating the system and the organization to understand how the failure transpired. This allows processes to be altered to reduce future risk. That's right, learning reduces risk.

At the pace we move, it is undeniable that your organization will develop intelligent questions regarding a failure that were missed immediately after past failures. Those new questions are crucial to the development of your organization, but they become absolutely precious if you can travel back in time and ask those questions about past incidents. This is what data retention in monitoring buys you. The new processes and interrogation methods you learn during your postmortems leading up to this year's cyber-Thursday shopping traffic can now be applied to last year's cyber-Thursday shopping traffic. This often leads to fascinating and valuable learning that, you guessed it, reduces future risk.

Be articulate about what success looks like

The final piece of advice for a successful monitoring system is to be specific about what success looks like. Using a language to articulate what success looks like allows people to win. It is wholly disheartening to think you've done a good job and met expectations, and then learn the goalposts have moved or that you cannot articulate why you've been successful. The art of the SLI

(service-level indicator), SLO (service-level objective), and SLA (service-level agreement) reigns here. Almost every low-level, ad-hoc monitor and every high-level executive KPI (key performance indicator) can be articulated in terms of “service level.” Understanding the service your business provides and the levels at which you aim to deliver that service is the heart of monitoring.

SLIs are things that you have identified as directly related to the delivery of a service. SLOs are the goals you set for the team responsible for a given SLI. SLAs are SLOs with consequences, often financial. Though a slight oversimplification, think about it like this: What is important? What should it look like? What should I promise? For this, a good understanding of histograms can help.

FROM RUM TO RSM

Monitoring in the web world moved long ago from the slow, synthetic ping of a website to recording and analyzing every interaction with every user; synthetic monitoring of the web gave way to RUM (real user monitoring) at the turn of the century and no one looked back. As we build more smaller, decoupled services, we move into a realm of being responsible for servicing other small systems—these are what engineering SLOs are usually built around.

The days of average latency for an API request or a database interaction or a disk operation (or even a syscall!) are disappearing. RSM (real systems monitoring) is coming, and we will, just as with RUM, be recording and analyzing systems-level interactions—every one of them. Over the last decade increased systems observability

(such as the widely adopted DTrace and Linux's eBPF) and improvements in time-series databases (such as the first-class histogram storage in Circonus's IRONdb) have made it possible to deliver RSM. (For a detailed look at why histogram storage of data is different and, more importantly, relevant, see the review by Baron Schwartz, "Why Percentiles Don't Work the Way you think." <https://www.vividcortex.com/blog/why-percentiles-dont-work-the-way-you-think>.)

Related articles

➡ Black Box Debugging

James A. Whittaker, Herbert H. Thompson

It's all about what takes place at the boundary of an application.

<https://queue.acm.org/detail.cfm?id=966807>

➡ Statistics for Engineers

Heinrich Hartmann

Applying statistical techniques to operations data

<https://queue.acm.org/detail.cfm?id=2903468>

➡ Time, but Faster

Theo Schlossnagle

A computing adventure about time through the looking glass

<https://queue.acm.org/detail.cfm?id=3036398>

RSM allows you to look at actual system behavior comprehensively, accounting for the whole distribution of observed performance instead of the synthetically induced measurements that consistently misrepresent the experience of using the system.

The transition from synthetic web monitoring to RUM was seismic; expect nothing less from the impending transition to RSM.

DON'T DELAY

Today, with architectures dynamically shifting in size by the minute or hour and shifting in design by the day or the week, we need to step back and remember

that monitoring is about understanding the behavior of systems, and that systems need not be limited to

computers and software. A business is a complex system itself, including decoupled but connected subsystems of sales, marketing, engineering, finance, etc. Monitoring can be applied to all of these systems to measure important indicators and detect changes in overall systems behavior.

Monitoring can seem quite overwhelming. The most important thing to remember is that *perfect* should never be the enemy of *better*. DevOps enables highly iterative improvement within organizations. If you have no monitoring, get something; get anything. Something is better than nothing, and if you've embraced DevOps, you've already signed up for making it better over time.

Theo Schlossnagle *is a software engineer and serial entrepreneur. He currently leads Circonus, where he focuses on building distributed systems and scale to help people analyze the behavior of their distributed systems at scale over time.*

Copyright © 2017 held by owner/author. Publication rights licensed to ACM.