


solarwinds 



 eBOOK

Monitoring 201:

Moving beyond simplistic monitors and alerts to #MonitoringGlory

by Leon Adato

Table of Contents

1	INTRODUCTION	4
2	ABOUT THIS GUIDE	6
3	ABOUT THE AUTHOR	7
4	ABOUT OUR SPONSOR	8
	The Four Phases of Development: Create, Test, Test, and Test	9
5	BRAINSTORMING, NOT BLAME-STORMING	12
6	A CHILD'S GARDEN OF MONITORS	13
	Universal Monitoring Crimes	14
	Server-based Monitoring and Alerts	18
	Network-based Monitors and Alerts	24
7	BONUS	25
	Bandwidth Utilization	26
	Application-based Monitors and Alerts	27
	Virtualization-based Monitors and Alerts	31
	Database-based Monitors and Alerts	32
	Top 10 Queries by CPU	34
8	SHOW ME THE MONEY	35
	Remember the Bad Old Days?	35
9	AVOID THE RETRO ENCABULATOR	36
10	GATHER YOUR POSSE	37
	Revenue, Cost, Risk	38

Table of Contents-Continued

11	THE PLURAL OF ANECDOTE	39
	Service Monitoring Starts at Home	39
	Clearing Temp-Tation	39
	Flagging the Problem Children, then Expelling them from School	40
	Signal to Noise	40
	Repetitive Strain	41
	Nobody Wants to Hear "Oops" in the Operating Room	41
	The RX is Restarting Services	42
	The Right Information at the Right Time	42
	The Price of a Cup of Coffee	42
	The Little Things Add Up	43
12	THE COMPLETELY UNECESSARY SUMMARY	45
13	APPENDIX A: WHAT CAN GO WRONG WITH IIS?	45
	Crash vs Hang	46
14	DEDICATIONS	47

Introduction

A few years ago, I realized I was repeatedly encountering the same problem: a lack of basic knowledge about what monitoring is and how it works. Managers would buy one solution expecting it to perform multiple operations, only to become frustrated when it didn't. There was very little understanding of the alternatives to the specific solution that had been purchased, which created frustration among teams who had conflicting views about that solution.



So, I wrote "[Monitoring 101](#)," a comprehensive guide to the fundamentals of monitoring. This vendor-agnostic e-book described in simple terms how each of the handful (okay, more like two handfuls) of techniques - the ones that form the basis for 90% of all monitoring solutions - worked, along with why and when they should be used.

The guide has become a valuable resource for monitoring specialists to share with new monitoring specialists, managers, and colleagues in other teams who consume the results of monitoring data. It helps everyone understand what is happening under the hood, what the limitations are, and what other options are available.

That was a good start, but it's time to put class back in session.

In this installment, I want to tackle the next major issue that I've found in most organizations with regard to their monitoring: simplistic monitoring that doesn't really lead to meaningful action.

What I mean is that once people understand the basics of monitoring and have a reasonably sufficient software solution in place, they load up their devices and start collecting whatever data comes out of the box, even when that data doesn't tell a truly meaningful story. This illustrates that people maintain a high level of trust in the metrics their vendors collect.

(It should be noted that, in most cases, this trust is well placed. I know very few monitoring products that PURPOSELY collect incorrect or meaningless data. But I've met many monitoring engineers who cannot describe why a particular metric is useful or necessary.)

You don't want to collect the wrong metrics, but doing so is fairly innocuous. As long as the database or storage team isn't complaining about the amount of space being "wasted" by "useless data," which nobody ever does because nobody looks all that closely, your software will collect and process stuff that will never be used.

But then the alerts get turned on, and that's when people's time really gets wasted.

Do you know what is wrong with an alert that triggers when CPU utilization is over 90%? Everything. It says nothing about what is going wrong, or even if anything is going wrong. As a SysAdmin, if I see a server that is consistently running at 90% and keeping up with its workload, I call that "correctly sized." It is likely that you do, too.

But what you'd really like to know is when the number of jobs waiting for CPU is greater than the number of CPUs in the system while there is high CPU utilization that has persisted for a significant length of time. Better still, the alert should tell me what the top running processes are at the time of the trigger.

That tells a clear story about what is wrong and how to go fix it.

These are the issues this guide is designed to address.

About This Guide

This guide was written to provide an overview of monitoring techniques and concepts that help the reader get beyond the out-of-the-box options that are often generic, simplistic, and less useful than they want, need, or appear to be.

Despite how that sounds, that's not an indictment of monitoring software vendors. Shipping a product with complex, labyrinthine, or highly specified alerts and monitors isn't helpful for beginners. In fact, it creates the impression that the software is useless, or worse, that monitoring is useless.

So, it's understandable that your shiny new monitoring software sets you up with the basics. It's easier to comprehend what the alert logic is doing, especially if you are also shiny and new (either to IT in general or to monitoring specifically). The mistake is thinking that what you find after installation is the be-all and end-all of all monitoring. Trust me, it gets much much better, especially if you are willing to put in a little work and think carefully about the environment you are monitoring. Ideally, you narrow the scope of events and data that should be monitored so you get the best information that leads you to the most efficient fix.

If you find yourself a bit behind on monitoring concepts, we can recommend the comprehensive (and free) "[Monitoring 101](#)" guide. There's also a great guide that focuses specifically on monitoring automation, called "[It's Automation, Not Art](#)" that you may want to check out as a companion to this book.

This guide is completely tool- and vendor-agnostic. It won't describe how to collect metrics or set up alerts, for example. Instead the focus is on discerning what the correct input metrics are, and why they are effective.

While the author has decades of experience with a wide variety of monitoring solutions, those experiences were used only in the sense that they provided insight into general trends and techniques. Nothing in this document will require that you use software package XYZ to actually dive into advanced monitoring and alerting with the ideas and techniques presented.

But it is the author's hope that after reading this, and after you've dispensed with the basics, you'll look at the alerts and monitors in software package XYZ and say, "Okay! Let's put this thing to work!"

About The Author

Over the course of a career that has spanned over three decades and four countries, Leon Adato has, at various times, been an actor, electrician, carpenter, stage combat instructor, pest control technician, Sunday school teacher, and ASL interpreter. He also worked on computers from time to time.



Leon Adato
SolarWinds Head Geek

Leon got his start providing computer classes at a time when the two requirements to teach computers were to – and this is right out of the help wanted ad – “have a pulse and own a suit.” From there, he worked his way up the IT food chain from desktop support, server support, desktop environment standardization engineer, and from there into the wild and oddly satisfying world of systems monitoring, management, and automation. Along the way he also discovered a weird love for taking tests, and picked up an alphabet’s worth of certifications including WPCR, CNE, MCP, MCSE, MCSE+I, CCNA, and SCP.

Leon spent almost 20 years honing his monitoring skills at companies that ranged from big (National City Bank) to bigger (Cardinal Health®) to ludicrous (Nestle®), becoming proficient with a variety of tools and solutions along the way.

A user of SolarWinds® software since 2003, Leon attracted the attention of SolarWinds staff due to his participation on the [SolarWinds THWACK® forums](#), along with an impressive point total that was achieved via a mixture of helpful posts, UX sessions, participation in beta and RC testing, and abject whining.

It was about that time that Head Geek™ Patrick Hubbard noticed that Leon was long-winded to a fault, and that he lacked the good sense to be nervous in front of large audiences.

And so, in 2014, Leon was offered a position as Head Geek. Hooked by the job title alone, Leon made the leap from in-house monitoring wizard to spastic fast-talking technical evangelist, and he hasn’t looked back since (mostly because he’s also incredibly uncoordinated and would probably run into something.)

About Our Sponsor

The author has used SolarWinds tools since 2003, and here's what he's learned in that time:

First, SolarWinds is "Geek Built™." That means that geeks, including SysAdmins, engineers, and other IT professionals, produce solutions for other geeks. SolarWinds addresses real problems that geeks face every day at work. We're not designing solutions based on which buzzwords are getting the most play on social media. Instead, we spend a lot of time talking to people in the trenches to find out not only what they are thinking about in terms of problems, but also how they would like to see those problems addressed. That feedback becomes the list of features we build into the next version.

Second, it's modular. You don't need to get the whole suite in one monolithic installation. You can determine which functionality you need, and then get the modules that meet those needs. The modules will snap together under a common framework, and also integrate well with solutions from other vendors. Because real geeks know that you don't get to pick every single piece of software the company uses, and that, like a good mutt, heterogenous solutions are often the most robust and faithful allies you can have in the data center. The flipside of this is that each module is flexible. Each tool has a variety of "outside-the-box" actions you can take to get almost any job done.

Finally, and there's no way to dress this up, SolarWinds solutions are affordably priced. Especially when you consider the features in each module, and function profile. Is it free? Of course not. My kids' orthodontist likes getting paid regularly, too. But you are definitely getting enterprise-class solutions at SMB prices.

Head over to <http://www.solarwinds.com> for more detailed information on products and pricing. You can also download a free, unlimited (meaning you can load up as many devices as you want), 30-day demo of any (or all) of the SolarWinds' modules.

Pro Tip: There are also about two dozen free tools you can download over at <http://www.solarwinds.com/free-tools/>

THE FOUR PHASES OF DEVELOPMENT: CREATE, TEST, TEST, AND TEST

Hopefully there are a few examples in this guide that will get you excited, spark your creativity, or maybe just make you say, “I need to see that for myself.” Whatever your reason, diving right in and putting one of them into action in your production environment is never a good idea.

Not ever.

I’m serious.



Before we can begin any discussion that suggests creating new monitors and/or alerts, we need to take a moment and discuss proper testing.

I mentioned this in the “[Automation, Not Art](#)” e-book, but it’s worth repeating and even elaborating on here:

- » Understand the difference between the scope of an alert and the trigger conditions. Even if your monitoring solution doesn’t divide these out, “Operating System = Windows” is not an alert trigger, it’s a scoping statement. When testing, ratchet the scope as small as you can and expand it slowly. What you want to test is the trigger condition first.
- » To that end, identify your test machines first: Whether that is lab gear set aside for the purpose, or a few less-critical volunteers, set up your alert so that it only triggers for those machines.
- » Learn to use reverse thresholds: While your ultimate alert will check for CPU>90%, you probably want to avoid spiking the test machines repeatedly. Just turn that bracket around. CPU<90% is going to trigger a whole lot more reliably, at least we hope so.

- » Find the reset option: Closely related to the previous point, know how your monitoring tool resets an alert so it triggers again. You will likely be using that function a lot.
- » Verbose is your friend: Not at cocktail parties or a movie theater, but in this case, you want to have every possible means of understanding what is happening and when. If your tool supports its own logging, turn it on. Insert, "I'm starting the XYZ step now," and "I just completed the XYZ step" messages generously in your alert actions. It's tedious, but you'll be glad you did.
- » Eat your own dog food: If you were thinking you'd test by sending those alerts to the production team, think again. In fact, you aren't going to send it to any team. You're going to be getting those alerts yourself.
- » Serve the dog food in a very simple bowl: You really don't need to fire those alerts through email. All that does is create additional delays and pressure on your infrastructure, as well as run the risk of creating other problems if your alert kicks off 732 messages at the same time. Send the messages to a local log file, to the display, etc.
- » Share the dog food: Now you can share them with the team as part of a conversation. Yes, a conversation.
- » Embrace the conversation: This process is going to involve talking to other people. Setting up a new (and likely more sophisticated) monitor or alert is collaborative, because you and the folks who will live with the results day in and day out should agree on everything from base functionality to message formatting.
- » Set PHASE-ers to full: Once the monitor or alert is working on your test systems, plan on a phased approach. Using the same mechanism you did to limit the alert to just a couple of machines, you are going to widen the net a bit, maybe 10-20 systems. And you test again to observe the results in the wild. Then you expand out to 50 or so. Make sure both you and the recipients are comfortable with what they're seeing. Remember, by this point the team is receiving the regular alerts, but you should still be seeing the verbose messages I mentioned earlier. You should be reviewing with the team to make sure what you think is happening is what is really happening.

Following those guidelines, any automated response should have a high degree of success, or at least you'll catch bad monitoring and alerting before it does too much damage.

So, which example should you start with? Find the things that have the biggest bang for the least effort.

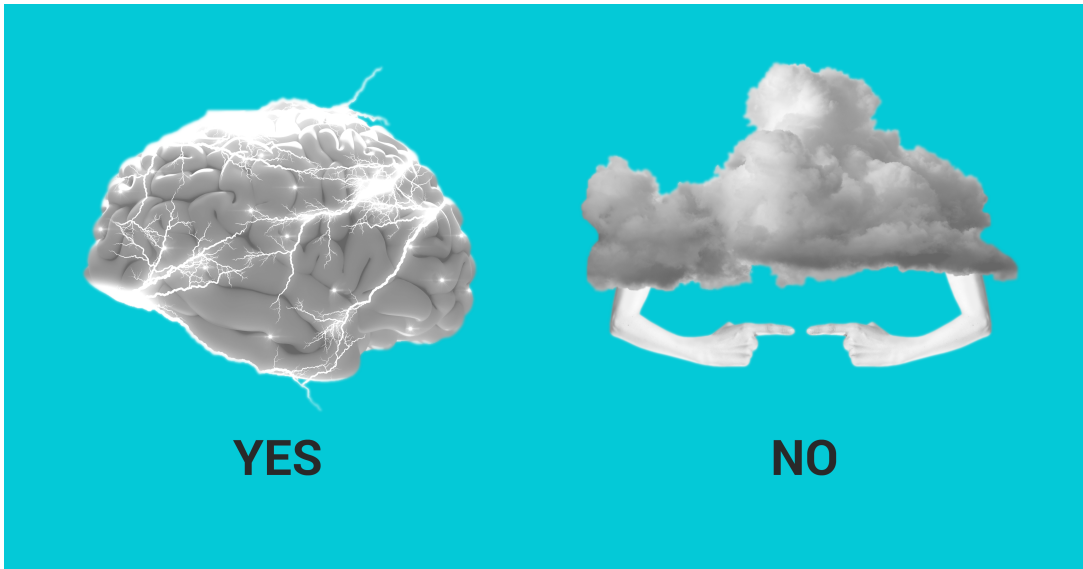
A great place to start is to look at your current help desk tickets. Look for monitoring-based alerts that have a high incidence of "mass close" operations (if your ticket system supports that feature) or where large numbers of the same type of ticket are closed at about the same time (within three minutes of each other).

These are likely the monitors and alerts that happen too often with no actionable result.

Another good place to find ideas for automation is the lunch room (whether physical or virtual – don't ignore your remote folks, they often know more about things that go bump in the night because their hours are more flexible). Listen to teams complain and see if any of those complaints are driven by system failures. If so, it may be an opportunity for a sophisticated alert to save the day.

Finally, don't plan too far ahead. As apprehensive as you may feel right now, after one or two solid, if small, successes, you will find that teams are seeking you out with suggestions about ways you can help.

Brainstorming, Not Blame-Storming



As we move into the core of this e-book, I wanted to give a quick overview. Each entry will break down into the following sections:

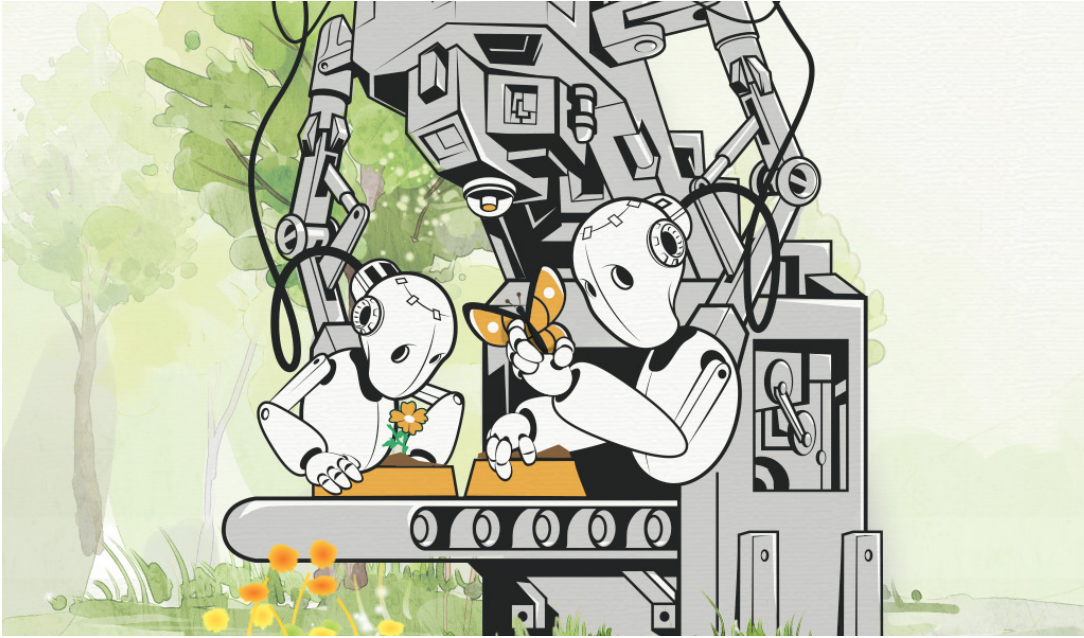
- » THE CULPRIT: Where I describe a typical/default monitor or alert that either comes pre-set in your monitoring solution, or is typically set up by novice monitoring engineers. But the key is that it is utterly ineffective.
- » THE CRIME: An explanation of why it is ineffective.
- » THE CORRECTION: Detailed information on how to make this type of alert meaningful to most IT professionals.

That said, I don't want to give the impression that anyone is at fault. People who are new to monitoring tools don't always know the sophistication that is available to them, and so they go for the lowest (or easiest) common denominator.

Meanwhile, monitoring vendors build simple and non-specific alerts on purpose, as a way to introduce users to the way alerts could work. They aren't meant to be used out of the box as much as they should be models upon which customers base their actual alerts.

If you find these types of alerts in your environment it means you are normal, your monitoring solution is normal, and everything is not going to hell in a handbasket. It just means you are about to read some pure monitoring awesomeness that you can use to make your environment (more) awesome too.

A Child's Garden Of Monitors



So here we are, at the heart of the matter. Below you will find examples of sophisticated monitors and alerts in several different categories.

Even with what you find below, you will most likely need to make slight alterations – salt to taste as the foodies say – because everyone's environment is just a little bit different.



However, I think that the information you'll find, which includes the reasoning behind why the original monitor was sub-optimal, and how the new design improves and makes things more usable – is worth it all by itself. It shows the thought process that should become second nature if you want to set up robust, effective, disciplined monitoring.

I want to start with the monitor that got this whole ball of wax started: [The Ultimate CPU Alert](#).

Monitoring CPU is simple, and alerting on high CPU is one of those things everyone sets up when they first tear the shrink wrap off their shiny new monitoring solution. But simply alerting on high CPU is, as I mentioned at the start of this e-book, wrong on so many levels.

But I'm getting ahead of myself. Let's break this down with a sample:

CPU ALERTS

The Culprit

Alerting when CPU Utilization is > xx%

The Crime

High CPU alone is rarely an indicator of any actual problem. In many cases, high CPU is proof that the system was sized correctly for the work that's being done...

(You'll find a full explanation in the Server-based Monitors and Alerts section.)

The Correction

The key is to collect three separate pieces of information:

1. The CPU utilization – CPU_UTIL
2. The number of CPUs (or cores) on the system – CPU_COUNT
3. The number of jobs waiting to be processed – CPU_QUEUE

(Again, you'll find a full explanation in the Server-based Monitors and Alerts section.)

With that seminal example out of the way, I have grouped the rest of my 201-level monitors by their general area of interest (server, application, network, etc).

UNIVERSAL MONITORING CRIMES

Fixed Thresholds

THE CULPRIT

Triggering any type of alert at a fixed value for a group of devices.

THE CRIME(S)

There are several crimes at play here:

» While general thresholds can be established, it is unlikely that every single device is going to adhere to the same one. Thus you end up with multiple versions of the same alert:

- CPU Alert for Windows® IIS™, DMZ
- CPU Alert for Windows IIS, core
- CPU Alert for Windows Exchange™ CAS
- CPU Alert for Windows Exchange MTA
- CPU Alert for those other Windows IIS, Special
- CPU Alert for those other Windows IIS, Special 2

- » Even for a single server, utilization varies from day to day, which is normal. So, you may have a server that usually runs at 50% CPU, but spikes to 95% at the end of the month. This is perfectly acceptable.

THE CORRECTION

There are two general solutions for this issue, depending on the sophistication of your monitoring solution:

- » Enable per-device thresholds. This may be done within the tool, or by using customizations. The point is that you should be able to have a specific threshold for each device. Then, your alert logic looks something like this (using CPU as the example metric):

```
IF
(CPU_THRESHOLD is not blank
AND %_CPU_Utilization > CPU_THRESHOLD)
OR
(CPU_THRESHOLD is blank
AND %_CPU_Utilization > default%)
```

In this way, machines that have a specific threshold trigger at the correct time, and those that do not get the default.

- » Baselines. These have the benefit of handling per-device alerts and also leverage the monitoring data you are already collecting. When using baseline data, your threshold is set to a % or value different than normal – with normal being determined by the baseline data for that time period.
- » If you use baselines (which I highly recommend you do, if your software supports them), you'll want to think through some of the edge cases. For example, if the condition you are trying to alert on is a constrained CPU and 2σ is still only 23%. That's not particularly helpful. So you may want to add a second condition (as we did above), such as:

```
WHERE(% CPU utilization > 2σ AND % CPU utilization > X%)
```

Lab, Test, and QA for your Monitoring

THE CULPRIT

No test or QA environment for monitoring.

THE CRIME

Many organizations set up a proof-of-concept environment for their monitoring solution, but once the production system is rolled out, they fail to have any type of lab, test, or QA system that is maintained on an ongoing basis.

THE CORRECTION

The hard truth that many organizations fail to recognize is that if your monitoring system is watching and alerting on mission critical systems within the enterprise, then it is mission critical itself. And if that is the case, then it needs to have the same level of rigor that would be given to other key systems, meaning one or more separate installations that serve roles such as:

- » TEST: an environment where patches and upgrades can be tested before attempting them in production
- » DEV: an environment that mirrors production in terms of software, but where monitors for new equipment, applications, reports, or alerts can be set up and tested before rolling those solutions to production
- » QA: an environment that mirrors the previous version of production, so that if issues are found in production, they can be double-checked to see whether the problem was introduced in the latest revision

I'm not saying that all of these environments are needed, but at least one is going to be highly beneficial to helping ensure that your monitoring is reliable.

Watching the Watcher

THE CULPRIT

What monitors the monitoring system?

THE CRIME

Having a tool or set of tools that monitor and alert on mission-critical systems, but having nothing in place to identify problems within the monitoring solution itself.

THE CORRECTION

The answer is pretty straightforward: Set up a separate instance of a monitoring solution that keeps track of the primary, or production, monitoring system. It can be another copy of the same tool or tools you are using in production, or a separate solution, such as open source, vendor-provided, etc.

Another option is to use the QA/lab/dev instance described earlier to also monitor the production toolset.

One more thing to consider for this subject is that monitoring services, availability, etc., for your production monitoring solution is good, but having a test that verifies end-to-end capability is better. The best way to make sure that everything is working optimally is to create a synthetic transaction that injects an event, helps ensure that an alert is generated all the way to the ticketing system – if your monitoring system goes that far – and then notifies you when that set of actions fails.

Instant Alerts

THE CULPRIT

Triggering alerts as soon as a condition is detected.

THE CRIME

In the vast majority of cases, the time for automated investigation and validation occurs the instant a problem is detected by the monitoring system.

The reasons why this is bad are legion. Monitoring systems are not perfect. Sometimes they pick up false positives. Sometimes problems appear for a moment and then just disappear. Some issues aren't really actionable unless they've persisted for a period of time. The list goes on.

THE CORRECTION

Except for a narrow category of alerts, a delay should be included in the trigger logic. So, the CPU alert mentioned earlier would need to have all of the specified conditions persist for, say, 10 minutes, before any action would be needed. Anything less represents a momentary spike in activity, from which the server could easily recover. Spikes lasting more than 10 minutes indicate a need for more direct intervention.

To set these trigger delays correctly, it is important to understand the interaction of your monitoring system's polling cycle, trigger alert check cycle, and the delay you put in place. This is described in greater detail in [this video](#).

One such exception to this rule are message-based alerts. These include input from traps, syslog, log files, and the like. Unless your threshold is the quantity of the same message over a period of time, the instant you receive the message is the instant you should act.

Example: When you receive a configuration change trap from a network device, you should immediately respond by pulling the new configuration, comparing to the previous iteration, and then acting appropriately if they are meaningfully different.

Flapping/Sawtooth Alerts

THE CULPRIT

Alert definitions that create multiple notifications for repeated events.

THE CRIME

As described in "[Monitoring 101](#)", when an alert repeatedly triggers (a device that keeps rebooting itself, a disk drive that hovers on the edge of full, and processes keep deleting/creating temporary files so that one moment it's over threshold, the next it's below), that condition is known as flapping or sawtoothing.

The crime should be obvious. Even if the threshold is momentarily resolving and then re-triggering, the problem never really reversed.

THE CORRECTION

Again, as mentioned in “Monitoring 101,” there are a few possible corrections, depending on what is supported by your monitoring solution and which suits the specific situation:

- » Suppress events within a window. Some software can ignore duplicate events during a period of time.
- » As mentioned in the previous scenario, you can add a delay before triggering.
- » Do not cut a new alert until the original has reset. Sophisticated monitoring tools will wait for a reset event before triggering a new alert of the same kind. This reset typically looks like the alert trigger, but in reverse (if the alert is > 90%, the reset might be < 90%). Better tools will let you code the reset rules separately so that you might trigger when disk > 90%, but it won't reset until it's < 80%. Even better, if you can add delays to both the alert trigger and the alert reset, you can trigger when the device is down for two minutes, but only reset after it has been up for 10 minutes.
- » Two-way communication with a ticket or alert management system. Best case would be for the monitoring system to communicate with the ticket and/or alert tracking system so you can never cut the same alert for the same device until a human has actively corrected the original problem and closed the ticket.

SERVER-BASED MONITORS AND ALERTS

CPU Alerts

THE CULPRIT

Alerting when CPU Utilization is > xx%.

THE CRIME

High CPU is, by itself, rarely an indicator of any actual problem. In many cases, high CPU alone is proof that the system was sized correctly for the work being done. The idea is that if the CPU is consistently high (but not pegged at 100%), and the system is keeping up with demand, then you are utilizing all of the hardware resources effectively.

THE CORRECTION

The key is to collect three separate pieces of information:

- » The CPU utilization (yes, you still need that) - CPU_UTIL
- » The number of CPUs (or cores) on the system - CPU_COUNT
- » The number of jobs waiting to be processed - CPU_QUEUE

Then the alert trigger should be set when:

```
(CPU_QUEUE > CPU_COUNT) AND (CPU_UTIL > xx%)
```

... for more than yy minutes.

In this situation, you know that the CPU is spiking AND the machine is not keeping up with demand. This is a clear sign that either a process has run wild, or it's time to upgrade your hardware.

In Windows machines, the CPU_QUEUE is going to be a performance monitor counter (Perfmon Counter). Look for it under the System category, listed as Processor Queue Length.

Meanwhile, on Linux® systems, CPU_QUEUE is called loadAverage15MinInt, and you can get to it with the following SNMP OID: 1.3.6.1.4.1.2021.10.1.5.3. Note that this is an integer, so you'll have to take this number and divide by 100 to get it to match up to the CPU_COUNT. If you want more granular monitoring, there's a five-minute value (1.3.6.1.4.1.2021.10.1.5.2), and even a one-minute value (1.3.6.1.4.1.2021.10.1.5.1).

Memory Alerts

THE CULPRIT

Alerting when memory utilization is > xx%.

THE CRIME

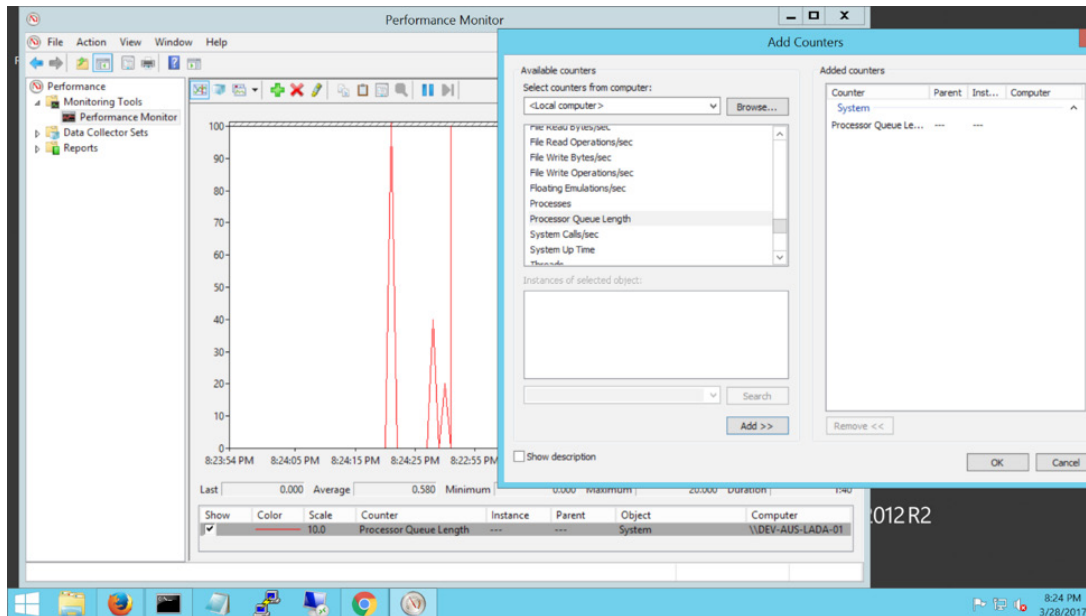
Memory alerts, like CPU, are one of the first things new monitoring engineers slam into place. This ends up causing recipients of those well-meaning alerts to consider monitoring to be little more than noise because saying that memory utilization is high, by itself, indicates very little.

THE CORRECTION

The key here is to see when memory utilization and paging (swapping memory to disk) are high at the same time. To do that, there are two additional elements on a Windows system that you need to consider: Page File Utilization(%) and Memory Pages Per Second.

So, the appropriate trigger formula would look something like this:

RAM Util% > 90% AND Page File Util% > 90% AND Memory Pages per Second > 25



On Linux systems, the formula is slightly more complex. You need

Real Memory % Used > 98% and SWAP Memory % used > 98%

But discerning a straightforward Real Memory and Swap Memory value is a little tricky. As outlined in Bob Ross' in-depth post here: <https://thwack.solarwinds.com/message/163840>, Linux RAM is calculated and displayed a few different ways. Without going too far into it, you can either calculate the real % memory used as:

$$1 - ((\text{memAvailReal} + \text{memBuffer} + \text{memCache}) / \text{memTotalReal})$$

OR (if you prefer, as described in the article linked above) as:

$$1 - (\text{memAvailReal} / \text{memTotalReal})$$

To save you the legwork, the SNMP OIDs for those values are:

1.3.6.1.4.1.2021.4.5.0 => memTotalReal

1.3.6.1.4.1.2021.4.6.0 => memAvailReal

1.3.6.1.4.1.2021.4.14.0 => memBuffer

1.3.6.1.4.1.2021.4.15.0 => memCache

Meanwhile, the other side of the equation is swap. For this, you need the OIDs:

.1.3.6.1.4.1.2021.4.3.0 => memTotalSwap

.1.3.6.1.4.1.2021.4.4.0 => memAvailSwap

That makes the formula for SwapMemory % Used pretty simple:

$$1 - (\text{memAvailSwap} / \text{memTotalSwap})$$

Disk Alert (with Clearing)

THE CULPRIT

Alerting when a disk is xx% full.

THE CRIME

As fellow Head Geek Thomas LaRock and I discuss in this episode of [SolarWinds Lab™](#), simplistic disk alerts don't help anyone. If you have a two TB disk, alerting when it is 90% used translates to having 204.8 Gb of disk spaces remaining.

That's quite a lot of space, and definitely not a reason to be concerned. To discern whether there's really a problem, you need more information.

THE CORRECTION

There are two main ways to look at disk space usage and alert effectively:

- » Current % usage and space remaining.
- » Disk usage as a delta over time.

CURRENT % USAGE AND SPACE REMAINING

As Tom and I discuss in the video above, a more accurate way of alerting on disk usage would be to look at both the % utilization in combination with the amount of space remaining. For example: `%util > 90% AND Space_Remaining < 50 Gb`, so you don't wake someone up in the middle of the night for something that could have waited until morning.

Another important thought is to test for the TOTAL disk space to further enhance the logic. As a simple example of this, if a disk is > 1Tb, then the logic I give above (`%util > 90% AND Space_Remaining < 50 Gb`) would be applicable. But if the disk was smaller (say 250Gb), then a different `Space_Remaining` threshold would be used.

The best part about this is that it can be set up within a single, albeit complex alert. This means you can reduce the overall management and maintenance of alert rules in the environment.

There is also the understanding that, especially with disk alerts, you often have time to turn the ship. Setting up a warning, rather than a critical, alert when, for example, `%util > 80 AND Space_Remaining < 100Gb` would allow you to use automation* to clear the temp directory, truncate log files, and possibly deal with the issue completely without human intervention. A second, more critical level could then be used when the situation was especially dire.

DISK USAGE AS A DELTA

Another way to look at disk alerts is to understand that it's not the disk usage at any given point in time which is a cause for concern, but the rate of consumption (or the delta over time). If the % of disk utilization is holding steady at 89% for a week, and then blips up to 91% that's probably

just a sign of normal usage. But if the usage spikes by 10% within a short period of time, that's a cause for concern even if the actual utilization is a jump from 55% to 65%.

Presuming your monitoring solution is able to track and trigger on such a condition, monitoring on a delta should, as with the previous scenario, be used in combination with other data points to determine the severity of the concern. Using the previous example, a 10% jump in 15 minutes might be a cause for concern but if that jump is going from 55% to 65% it certainly shouldn't be a reason to pull someone out of a meeting (ie: it's a warning, not a critical threshold). But a 10% jump from 89% to 99% definitely is.

* For more about automated responses to disk alerts (as well as many, many other conditions) please check out our ["Automation, Not Art"](#) e-book.

Monitoring for a Locked-out Monitoring Domain Account

THE CULPRIT

Monitoring solutions, especially those that use agentless collection techniques, often use domain accounts to connect to systems and pull statistics.

THE CRIME

If that account becomes locked, life for you as a monitoring engineer tends to get very ugly, very fast. While this isn't a crime per se, because monitoring solutions need to get their statistics somehow, agentless techniques are especially vulnerable. That's why this situation should be monitored and alerted on.

THE CORRECTION

There are two hurdles to overcome to mitigate this situation:

- » You can't monitor the account using the same system that will be impacted if the domain account gets locked.
- » You need a second domain account to check whether the main account used by monitoring has been locked.

That said, there is a very simple, script-based solution that uses Windows PowerShell™, which can run on the monitoring solution you use for testing, proof of concept, etc. You could run this script via a Windows task or other job scheduler, and set up your own notification method.

```
## Returns either 'True' or 'False' from the LockedOut property given the name of the  
User Logon Name
```

```
$is_locked = get-aduser <MyAccountName> -Properties * | Select-Object -exp LockedOut  
if ($is_locked -eq 'True')  
{  
Write-Host 'Statistic: 1'  
Write-Host 'message: Account Locked Out'}
```

```
}else{  
Write-Host `Statistic: 0`  
Write-Host `message: Account Not Locked`  
}
```

Monitor Windows Task Manager with SNMP

THE CULPRIT

Not being able to get current processes off a Windows-based system.

THE CRIME

Simple Network Management Protocol (SNMP) is an amazing thing. It is tight and efficient and, as the name implies, simple. It is ubiquitous across network devices, Windows, Linux, and even IoT devices, such as your internet-connected coffee pot.

It is also old and somewhat archaic, especially in the face of newer, albeit less efficient or elegant monitoring options such as WMI, IPSLA, or API-based techniques.

Some monitoring tools are limited to SNMP, though. If that's the case, you may believe you cannot get a list of the top running processes on a Windows system, whether for display purposes or to inject insight into an alert.

This is not the case.

THE CORRECTION

First, here are the SNMP Object IDs (OIDs) you'll need:

- » hrSWRunName 1.3.6.1.2.1.25.4.2.1.2
- » hrSWRunPerfCPU 1.3.6.1.2.1.25.5.1.1.1
- » hrSWRunPerfMem 1.3.6.1.2.1.25.5.1.1.2

To make this work, you will need to:

- » Set up each of those OIDs using a Get Table operation
- » Pull your label names from the hrSWRunName data set
- » Join all of the data sets using the IndexID found in each of the table sets
- » Convert the CPU value into a percent by using the following formula:

```
truncate(((hrSWRunPerfCPU)/columnsum( {hrSWRunPerfCPU}))*100,0)
```

You'll have a display that looks like this:

ESX Host Details - ● EASTESX01A - 🏠 Custom

<<

- 🏠 SUMMARY
- 📁 STORAGE
- 📦 ASSET INVENTORY
- 🖥️ VIRTUALIZATION SUMMARY
- 🏠 **CUSTOM**
- ⊕ ADD TAB

Task Manager EDIT HELP

HRSWRUNNAME - LABEL	PCTCPU	HRSWRUNPERFMEM
busybox	0	776
busybox	0	776
cimsip	0	1560
dcdb	0	2724
dcui	0	16124
fdm	1	17356
hostd	9	69388
init	0	2296
net-cdp	0	1288
net-lacp	1	2732
net-lbt	2	2808

NETWORK-BASED MONITORS AND ALERTS

Errors and Discards

THE CULPRIT

Packet error and/or discarded packets alerts.

THE CRIME

Both of these are simple counts, which could rise dramatically, or slowly, but could end up being a large number that signifies nothing important.

THE CORRECTION

Errors (and discards) are a report of how many packets had issues. The number of errors is relative to the total number of packets sent and/or received. If 100 of 10,000 packets result in errors (1%) that is something you want to take a look at, whereas if 10,000 packets of 10,000,000 error (0.1%) then you've got little to worry about, even though the total number of packets is higher.

Is 1% the right threshold? Maybe or maybe not. The point is to have the ability to measure against the overall packet count.

Bonus!

Josh Biggley - SolarWinds expert, THWACK MVP, and all around great guy – originally contributed this crime. Never one to leave people wondering how to actually implement a solution, he’s provided a SQL query to get the exact measurements described above. Some aspects of this example are specific to the Solarwinds database (especially field names), but even so, this should give you a general idea of how such an alert trigger would be configured, which you can modify to fit your monitoring solution:

```

SELECT Interfaces.FullName, Interfaces.InterfaceID FROM Interfaces
WITH (NOLOCK)
JOIN
(
SELECT
    ie.[InterfaceID]
    ,CASE WHEN
        ie.[In _ Errors]+ISNULL(it.[In _ TotalPkts],0)=0
        THEN 0
        ELSE (((In _ Errors))/(ie.[In _ Errors]+ISNULL(it.[In _ TotalPkts],0))) * 100
        END ReceivePercentErrors
    ,CASE WHEN
        ie.[Out _ Errors]+ISNULL(it.[Out _ TotalPkts],0)=0
        THEN 0
        ELSE (((Out _ Errors))/(ie.[Out _ Errors]+ISNULL(it.[Out _ TotalPkts],0))) * 100
        END TransmitPercentErrors
FROM [dbo].[InterfaceErrors _ Detail] ie WITH (NOLOCK)
LEFT JOIN [dbo].[InterfaceTraffic _ Detail] it WITH (NOLOCK) ON ie.InterfaceID=it.InterfaceID
AND ie.[DateTime]=it.[DateTime]
WHERE
(
    In _ Errors > 0
    OR Out _ Errors > 0
)
-- Events from the Last 7 Minutes to match interface polling interval
AND ie.DateTime >= DATEADD(SS, -420, GETDATE())
) AS T1 ON T1.InterfaceID = interfaces.InterfaceID
--Not required unless filtering on a node custom property
JOIN NodesData nd WITH (NOLOCK) ON interfaces.NodeID = nd.NodeID
JOIN NodesCustomProperties ncp WITH (NOLOCK) ON nd.nodeid = ncp.nodeid
WHERE
([T1].[TransmitPercentErrors] >= 1
OR [T1].[ReceivePercentErrors] >= 1)

```

BANDWIDTH UTILIZATION

THE CULPRIT

Alerting when bandwidth is over a particular threshold that is not 100%.

THE CRIME

Like CPU, monitoring raw bandwidth tells you nothing about what is actually going wrong, or if anything is actually going wrong, and often causes non-network people to panic unnecessarily.

Running consistently at 97% bandwidth is, by itself, a sign that you purchased EXACTLY the right amount of bandwidth for your needs.

THE CORRECTION

Correcting this is actually very simple. All you need to do is add in response time. Because when bandwidth is over a certain level and your response time through that same interface is high, that indicates a bottleneck that needs to be addressed.

Once again, Josh Biggley has jumped in with some specifics. The overall trigger would look something like this:

```

WHERE
Bandwidth > (Bandwidth_Crit OR 92%) AND
Response Time > the value for 2σ
MUST EXIST for 21 minutes or more

And the SQL for such a beast might look like this (again, this is specific to SolarWinds but you
can get the gist regardless)

SELECT Interfaces.FullName, Interfaces.InterfaceID FROM Interfaces
INNER JOIN Nodes ON (Nodes.NodeID = Interfaces.NodeID)
INNER JOIN NodesThresholds NodesThresholdsResponseTime_NodesThresholds ON
    Nodes.NodeID = NodesThresholdsResponseTime_NodesThresholds.InstanceId
    AND NodesThresholdsResponseTime_NodesThresholds.Name = 'Nodes.Stats.ResponseTime')
WHERE
(
    (Nodes.OwnerGroup = 'NETWORK') AND
    (Nodes.Prod_State = 'PROD') AND
    (Nodes.n_mute <> 1) AND
    (Interfaces.i_mute <> 1) AND
    (Interfaces.InterfaceCategory = 'WAN')
    AND
    (
        (Interfaces.Bandwidth_Crit IS NOT NULL)
        AND (Nodes.ResponseTime >= NodesThresholdsResponseTime_NodesThresholds.
Level1Value)
        AND (Interfaces.InPercentUtil >= Interfaces.Bandwidth_Crit))
    OR
    (
        (Interfaces.Bandwidth_Crit IS NULL)
        AND (Nodes.ResponseTime >= NodesThresholdsResponseTime_NodesThresholds.

```

```
Level1Value)
    AND(Interfaces.InPercentUtil >= 92)
)
)
)
```

Another way to consider the question (not the problem, but the question that leads to identifying a situation as a problem in the first place), is to use NetFlow to show how that bandwidth is being utilized.

Consider the following two scenarios:

- » Bandwidth on the interface that feeds the main application database is running at 97% consistently. Response time through that interface is within normal levels. NetFlow shows the majority of the traffic is on port 1433 (the default MS-SQL port).
- » Bandwidth on the office router runs at about 60%. Response time through that interface is higher than you'd expect. NetFlow shows the traffic is a mix of Netflix® (50% of usage), MSNBC® (20%), social media (20%), and traffic to the main office (10%).
 - Using just bandwidth, you'd create an alert for the first scenario but not the second
 - Using the two metrics I mentioned earlier (bandwidth and response time), you still wouldn't flag the second scenario, but you also wouldn't have marked the first one

NetFlow would allow you to identify that even though bandwidth utilization is perfectly fine, the combination of slow response time and the specific uses of that bandwidth are cause for concern.

APPLICATION-BASED MONITORS AND ALERTS

IIS Application Pool Restarts

THE CULPRIT

Clearing the IIS Application Pool.

THE CRIME

Everyone who has worked with Microsoft® IIS for more than 10 minutes knows that the first thing you do when a site stops responding is to restart the application pool. But far fewer people understand why that fixes things, and therefore, which use cases will not be resolved with this action.

THE CORRECTION

Really this boils down to knowing when (as well as how) to restart the application pool depending on the scenario. Because a lot of this hinges on a solid understanding of IIS's architecture, we've included detailed information in [Appendix A](#). Once you understand that, the following will make more (if not perfect) sense:

In the case of a crash or hang, http.sys is maintaining the user connections at least until the timeout is reached. Recycling the application pool means instantiating a new worker process that can then pick up those lingering connections and begin processing them again. This avoids the need to perform more aggressive techniques, such as restarting IIS, or worse, the server itself.

In load-balanced environments, many systems administrators first remove a misbehaving server from the group of servers so that no new connections can be made while the issue is being resolved.

In general, this is the right thing to do, but when recycling an application pool, it actually slows down the process.

On a live server, the process of recycling the application pool goes like this:

- » The new worker process is started
- » Incoming traffic is sent to the new process
- » That process is registered as functioning
- » Once the new process is registered, the old process is shut down

By disconnecting a server from its load balance group, the new process isn't registered as functioning, and the old process doesn't shut down until the timeout is reached.

However, this directly leads to the next issue:

Restarting IIS

THE CULPRIT

Restarting IIS when there are still live connections.

THE CRIME

Restarting IIS when there are still active connections guarantees that new incoming users are going to see an error. When there is only a single web server, this is unavoidable. But in the case of a load-balanced web environment, it's unnecessarily disruptive.

THE CORRECTION

This is the exact reverse of the final point made earlier. If restarting the application pool fails to resolve the issue, then the next logical step is to restart the IIS service as a whole.

But before doing so, make sure the server has been removed from the load-balancing pool so that newly arriving users will not accidentally connect to the misbehaving server mid-restart. Once IIS is back up and verified, it can be added back to the load balance group.

Service Restarts

THE CULPRIT

Simplistic “service down” alerts.

THE CRIME

In this era of robust monitoring tools, there is no excuse for having an alert that simply tells you, “Service XYZ crashed.” Doing so means that, in the case of mission critical applications, you will either keep NOC staff busy but bored with tedious and meaningless actions. Or, in the absence of 24-hour coverage, you will have to wake critical staff to take an action that could have been performed by a well-trained chimpanzee.

THE CORRECTION

Any monitoring tool that can detect an error with a service – whether it is down, consuming too much (or too little) RAM, spiking CPU, or some other easily detected issue – should also be able to restart that service.

More to the point, the automatic restart should be followed with a second check to help ensure the service is running correctly. If not, additional escalation should take place.

As discussed in the [“Automation, Not Art”](#) e-book, the inclusion of automated responses, even on smaller but more frequent events like this, can save organizations thousands of dollars in staff time, lost opportunity cost, and even customer interactions.

So, the proper flow is:

- » Monitor the service
- » Service displays an error
- » Monitoring application automatically restarts the service
- » Short wait cycle while the service restarts and stabilizes
- » Service is re-checked
 - If the service continues to display problems, an escalated action is taken (critical ticket, page out to staff, etc.)

- If service is stable, non-critical ticket is logged so staff can review at a convenient time

Orphaned User Sessions

THE CULPRIT

Ignoring orphaned user sessions or waiting for the server to just run out of connections.

THE CRIME

Somewhere in the murky past, the first computer went online and became Node 1 in the vast network we now call the internet. The next thing that probably happened, mere seconds later, was that the first user forgot to log off their session and left it hanging.

For any system that supports remote connections—whether it's in the form of telnet/ssh^{*}, drive mappings, or RDP sessions—having the ability to monitor and manage remote connection user sessions can make running weekly, if not daily, restarts unnecessary. Or at least much smoother.

THE CORRECTION

For Linux, use the `W` command to list out remote connections with the connection duration and idle time. Then use the `Who` command to discover the session ID, or with greater granularity, by remotely running `netstat -tnpa | grep ESTABLISHED.*sshd`. Once you have the process ID, you can kill it.

For Windows, the easiest way to get the information you need is to use the PowerShell `quser` command. While this is not the simplest command to get the hang of, doing an internet search for “`quser get-loggedonuser powershell`” should yield a number of great scripts where people have done the work for you.

DNS Cache issues

THE CULPRIT

Failing to monitor the DNS Cache, or respond appropriately when there are issues.

THE CRIME

At times, a server or application will misbehave because it can't contact an external system. This misbehavior may be due to corruption in the DNS cache, or because the remote system has moved but the local DNS hasn't updated.

THE CORRECTION

Detecting issues with DNS is usually done using synthetic transactions that test DNS connectivity and validity on a regular basis. So, a check would be set up to query a specific DNS server for the name and IP of a known device. The monitor then checks the results against the expected results.

Fixing the issue is as simple as clearing the cache and letting the DNS server refresh its entries.

In Windows, use the command `ipconfig /flushdns`.

In Linux, the command varies from one distribution to another, so it's possible that `sudo /etc/init.d/nscd restart` will do the trick, or `/etc/init.d/dns-clean`, or perhaps another command. Check what the correct commands are for your version.

VIRTUALIZATION-BASED MONITORS AND ALERTS

Virtual Machines and CPU

THE CULPRIT

Monitoring CPU utilization on virtual servers.

THE CRIME

Monitoring a virtual machine as if it is a physical server has its advantages, but one of the biggest fake-outs is CPU. Just because a virtual machine believes the CPU is experiencing high utilization has little to no bearing on whether the physical resource is at capacity or not.

Conversely, even with the virtual machine's operating system reporting low CPU utilization, a problem at the host level could cause the VM to operate (from a processing standpoint) at sub-optimal capacity.

THE CORRECTION

There are two values that need to be discussed before the correct monitor and alert can be given:

CPU Ready Time (RDY%)

CPU Ready Time describes the condition in which the VM has work to do (called a "ready to run" state), but has to wait for the hypervisor to schedule that work on one or more of the physical CPUs. This is typically seen when a physical host is over-subscribed (too many VMs) or where a larger VM with an SMP application (like SQL Server®) is on the same host with a number of smaller VMs.

NOTE: On Microsoft Hyper-V® environments, the equivalent value is called "Wait Time Per Dispatch."

Co-Stop (%CSTP)

Co-Stop is the amount of time an SMP virtual machine was ready to run, but incurred delay due to co-vCPU (virtual CPU) scheduling contention. In a multi-vCPU VM, this metric indicates either:

- » The amount of additional time after first vCPU is available until other vCPUs are ready for the job that needs to be processed, or

» Any time the vCPU is stopped because of scheduling issues

NOTE: On Microsoft Hyper-V environments, the equivalent value is obtained using a Performance Monitoring (PerfMon) counter called Inter-processor interrupts /sec.

While it sounds like CPU Ready and Co-Stop measure the same thing, the key difference to remember is that Co-Stop is a metric specifically meant to measure SMP VMs, (the value would be zero on a single CPU VM), whereas CPU Ready applies to any VM in the system.

Putting this all together, a useful alert threshold would look for:

CPU Ready > (10% * <vCPU count>)

OR

Co-Stop > 3% for an extended period of time

Or for Hyper-V systems, it would be:

Wait Time > 40ms

OR

Inter-processor interrupts /sec > (20 * <number of vCPUs>) for an extended period of time

DATABASE-BASED MONITORS AND ALERTS

Page Life Expectancy / Buffer Cache Hit Ratio

THE CULPRIT

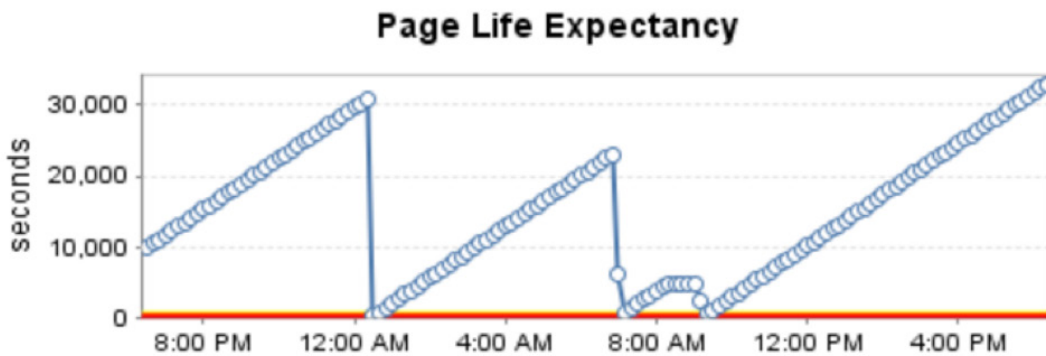
Using page life expectancy or buffer cache hit ratio indicators of database performance.

THE CRIME

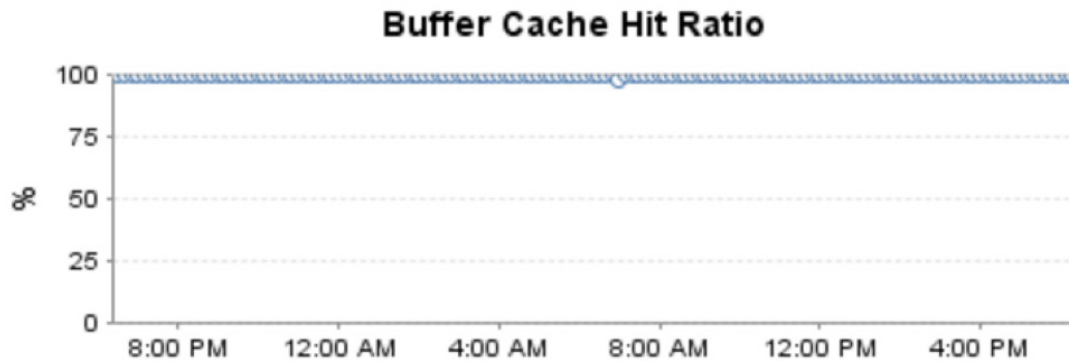
By itself, Page Life Expectancy – the amount of time a page of data remains in memory without anything explicitly requesting that page of data – appears to be a very important statistic. It certainly was, once upon a time, when servers sporting 4Gb of RAM were the Big Machines on Campus (BMoC).

According to experts (SolarWinds Head Geek Thomas “SQLRockstar” LaRock among them), the goal is to have every page of data remain in memory forever because it is so useful. A low number (under 300) indicates that there’s not enough memory on the system and data is being paged out frequently.

Based on that, a graph like this could be construed as a cause for concern:



According to the Database Performance Analyzer (DPA) knowledge base, buffer cache hit ratio (BCHR) is “the rate at which SQL Server finds the data blocks it needs in memory rather than having to read from disk for this instance.” Therefore, BCHR alone is not very meaningful. Because of read-ahead technology, your BHCR can be 100% throughout a crisis because the engine will know it needs additional pages and grab them, and they will be “hit” in the cache. In other words, BHCR can tell you what happened, but it can’t really predict what is about to happen.



THE CORRECTION

As described by SolarWinds Head Geek Thomas (SQLRockStar) LaRock in his blog post here, (<https://thomaslarock.com/2012/05/are-you-using-the-right-sql-server-performance-metrics/>)

“What you really want to see is the rate at which your pages are being cycled through the buffer pool. I usually look for rates around 20MB/sec as a baseline. Why that number? Because if I have a typical server with 56GB of RAM available for the buffer pool (thank you, locked pages in memory) and I want to keep my pages around for an hour or so (3600 seconds), then I come up with 56000MB/3600sec, or about 15.5 MB/sec. That’s why I look for a sustained rate of about 20, on average, and if I have a spike upward from there I know I am having memory pressure (pressure that might otherwise fail to be seen if I only examine the PLE counter).”

Based on the three graphs above, the changes in page life expectancy are due to some other issue, possibly changes in workload during various times of the day. The buffer cache is working fine, and there was one little blip at 8 a.m., but it didn't hit the 20Mb/sec threshold. So, it's really nothing to worry about and shouldn't have triggered an alert anyway.

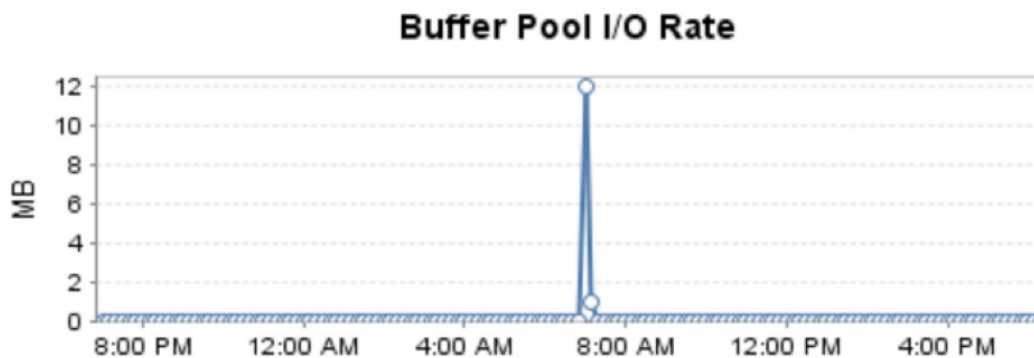
TOP 10 QUERIES BY CPU

THE CULPRIT

Alerting based on a "top 10 queries, sorted by CPU utilization" list.

THE CRIME

"Top 10 queries by CPU" is just about useless. Sure, it's become an industry standard, but it is virtually meaningless in terms of actionable information. If the top 10 queries by CPU are processing data efficiently, and are, in fact, the top 10 most run or most important queries, then everything on the server is running as intended.



THE CORRECTION

With database performance, the real story lies in locking, blocking, and waits. Some quick definitions:

- » Locking is when a session holds a lock on a resource and other sessions attempt to acquire conflicting locks on the same resource. The second session will wait on one of the LCK_M wait types, and depending on conditions, severe performance degradation can occur.
- » Blocking is the result of a separate transaction trying to access a resource that is locked.
- » Wait occurs whenever a thread or session has to wait for something before executing. It could be that a resource is waiting for a locked resource to become available, or data to be loaded into the buffer cache, or a number of other conditions.

The upshot of all this is that by shifting the focus of your monitoring and alerting from "top queries" to queries that encounter the largest amount of wait in a given period of time, you'll be far more likely to find, address, and resolve performance issues.

Show Me the Money



Monitoring improvements, especially the kind that we've been discussing here, takes time. It takes time to dream up, develop, and test. It takes time to test some more, and one more time after that, because we're professionals here and we know how things go. It also takes time to deploy. And, please excuse the cliché, all of that time translates into money for the business.

So, smart monitoring specialists need to take the accountants into account. While this guide can't tell you everything you need to do to satisfy the number crunchers at your company, this list of suggestions should at least get you started.

REMEMBER THE BAD OLD DAYS?

You need to, at least from a numbers perspective. Make sure you have data on the ticket counts before you install a new automation system. This will allow you to say things like, "Before monitoring, we were generating 800 systems-related tickets a month, with approximately 200 for interface issues, 400 for system outages, 150 for service failures, and 250 for assorted application issues. After implementing these improved alerts, those tickets dropped to, etc..."

Avoid the Retro Encabulator

Watch this [video](#). This, my friend, is what you sound like to everyone else. It's especially what you sound like to the business leaders at your company. They may nod appreciably, but there's a good chance that your pet iguana understands more of your conversation than they do.



So, in the name of getting what you want, skip it. All of it. Instead, put things into terms they care about. You have the ticket data from the previous step, right? Now turn it into dollars.

The algorithm isn't difficult. Set an estimated amount of time to resolve each of the ticket types that you listed in the previous steps. Something like:

- » Interface issue - .75 hour
- » System outage - 1 hour
- » Service failure - 1.5 hour, etc.

Now ask your number crunchers what the average total loaded cost of an employee is. This is the hourly rate for an employee that factors in everything about them, including their portion of the heat and electrical bill.

Now multiply the <total loaded cost> x <time per issue> x number of tickets for that issue.

This will give you the total cost for that issue for that period of time. If you do this for the before and after phases, you know how much you've saved the company, just by adding or improving monitoring and alerting.

To help you out, I've included a whole section of real-life examples near the end of this e-book.

Gather Your Posse



Despite our suspicions to the contrary, your typical business leader's day does not comprise three hours spent skimming *The Wall Street Journal*®, followed by a round of golf, martinis, and one meeting with you, where they listen perfunctorily and simply say “no” at the end.

Rather, their day is an endless series of meetings where the person at the front of the room declares how much they know (see “The Retro-encabulator” in the previous section) followed by some version of “trust me, this is important.” Inevitably, they request an amount of money that the business leader suspects is three times what is needed to protect against eventual budget cuts.

One of the best ways to attract the kind of attention you want from important decision-makers in your company is to provide them with solid numbers. Another way is to show your results in the form of end-user testimonials. When coupled with numbers, positive descriptions of experiences before and after installing automation provided by people from various departments reinforce the idea that the effort required to implement automation is worth the investment.

Doing this also has the secondary benefit of reinforcing the message that investing in the monitoring solution was also a good decision, and that further investments will have similar positive outcomes.

REVENUE, COST, RISK

During the 2015 THWACKcamp™ session, “Buy Me a Pony,” I sat down with the SolarWinds CTO to discuss the drivers that help executives make decisions. He boiled it down to three things:

- » Increasing revenue
- » Reducing cost
- » Avoiding risk

If your project, software, initiative, etc., can't speak to one of those things, it's simply not going to be a priority for them.

The good news is that effective monitoring does at least two of those things. It helps to reduce costs by catching issues sooner in the failure process, potentially before they spiral out of control, which clearly helps reduce business expenses.

Monitoring also helps avoid risk. Conversations about risk mitigation and avoidance frequently focus on ways for teams to predict, and then circumvent, potential failures that could affect a system, application, or service. Monitoring comes into play when you've come down from that philosophical mountain and accepted that some failures are simply unavoidable. So, in this case, risk monitoring helps you avoid the downstream consequences of failure by detecting and responding to it as soon as possible.

The Plural of Anecdote

If you are just getting started with monitoring (or improving what was in place when you got into this job), it may still be hard to imagine that a simple script could have that big of an impact on your environment. To help with that, I've collected a few war stories from friends and colleagues. Use these as you start to socialize the idea of monitoring automation in your company, at least until you you're your own stories to tell.

It's important to understand that I didn't include these stories because the monitoring was particularly complex or the savings were in the stratosphere. Quite the opposite, in fact. The solutions featured here are extremely modest and easy to accomplish, and the savings are meaningful but not in the millions of dollars. Trust me when I say that once you get started with monitoring, those million-dollar opportunities will present themselves. But the lesson here is that smaller efforts have the ability to yield measurable returns.

SERVICE MONITORING STARTS AT HOME

Shared by Josh Biggley, Monitoring Engineer for Cardinal Health:

"Since our monitoring is heavily dependent on SNMP, we have an alert in place that checks whether that service is actually responding on Windows systems. If not, we automatically restart the service. From its inception until October 2014, we were auto-restarting the SNMP service and created an average of 1.98 incidents per day. In October 2014, the Windows team requested that we remove the logic to restart SNMP. Between October 11, 2014 and January 12, 2015, we averaged 6.35 incidents per day, an increase of 4.37 incidents per day.

Even if the total incident handling time was only 15 minutes for all parties involved, removing that logic added an extra 65.5 minutes of work per day, or nearly 400 hours per year. That's 1/5th of an FTE. As you can imagine, when we highlighted these numbers to the team, they asked us to re-enable that SNMP restart logic."

CLEARING TEMP-TATION

Shared by Josh Biggley, monitoring engineer at Cardinal Health:

"While we developed some very sophisticated alerts for "disk full" monitoring, we still were creating over 700 tickets per month for this one event type, the largest volume of tickets for a single event in the enterprise. As we discussed it with the server teams, they pointed out that in the majority of cases, clearing the temp directory was all they had to do to resolve the issue.

So, we tested and rolled out a solution to do that automatically. We always open a ticket, but if the disk-clearing script succeeds, we update the ticket as "deferred" rather than "open" and the support team isn't paged out.

To give you a sense of how successful that is, we deferred 408 tickets between August 8 and August 31. We presume that responding to a “disk full” event takes staff an average of 15 minutes to manage. But even with that minimal time, we’re talking about 17.7 events, or 4.4 hours per day.

Automation is saving us half of one staff person each and every day for just this one event type alone!”

FLAGGING THE PROBLEM CHILDREN, THEN EXPELLING THEM FROM SCHOOL

Shared by Peter Monaghan, CBCP, SCP, ITIL ver.3

“We’ve set up similar scripting actions regarding disk space alerts and Windows Services unexpected stops. Since we don’t have a 24x-7 NOC, we have introduced “repeated” SolarWinds alerts for Tier 1 IT Services so that if one of those services fails after hours, scripts are initiated upon alert generation to automatically restore services. If services aren’t restored, alerts are generated every 30-45 minutes (depending on service) until either the script or manual intervention restores it.

Once I set up monitoring for these services, I was able to focus on the repeat offenders and ultimately reduce all chronic alerts and outages by over 70% for all Tier 1-3 IT Services.”

SIGNAL TO NOISE

Shared by Rick Schroeder, Network Administrator

“We experience approximately four unscheduled WAN outages per month. Our affected sites range from five employees to three hundred.

Using monitoring automation, we reduced the amount of downtime by 15 to 60 minutes per site, per incident simply through increased visibility by getting the right information to the right teams faster, so they could respond. That included providing near real-time information on outages of the WAN vendor so they could begin verifying and testing, perhaps bringing in last-mile providers, intermediate providers, or rolling a truck and technician to the site.

Some of the outages were six hours or more – if the issue was caused by backhoe fade or trees falling on aerial WAN connections.

Depending on the number of users affected, we calculated our savings at \$86,400 per year for our larger sites, and \$375,000 for outages at our data center, which only occurred once every three years or so, but in addition to being costly, it’s also very visible.

And all of that doesn’t consider the intangible costs for customers lost and impressions left.”

REPETITIVE STRAIN

Shared by Will Luther, Analyst Network Operations, GVTC

"A few years back, before we had any automation in place, I had to manually go through and back up the configs for all of our devices (starting with 200+). So, every week I would manually connect to EVERY device, paste my commands in, and back up the configs. This process took me approximately two full workdays to complete, every week.

As our network grew, my notepad of commands to be copied and pasted evolved into a collection of "expect" scripts. This was a major leap forward, and cut down the time to back up significantly. While it had previously taken me roughly two full days to perform all of the device backups, I was now able to back up those same devices in less than a day.

Our network eventually grew to be 1000+ devices, and even with those expect scripts, it started to take longer, yet again. Additionally, in this time frame, we had several devices die. And, while I was able to eventually provide the backed up configs, it was still cumbersome and time consuming.

I finally got the approval to get an automated configuration management system.

That was the answer, and the last evolution of our config management system. Now, I spend exactly 100% of zero days each week, backing up configs. The configs are easy to find, which makes re-configuring devices painless.

So automation has recovered 824 hours of staff time per year, not counting the faster resolutions when there is a device failure."

NOBODY WANTS TO HEAR "OOPS" IN THE OPERATING ROOM

Shared by Cahunt

"At one point, the hospital where I worked had an in-house developed EMR system that was distributed across several servers. The system would fail if a single service on any of those servers stopped unexpectedly. That application team's way of managing this was logging each server to check the service/process status one at a time until they found the stopped or hung service and then restart the servers in a specific order that would allow the service/process to restart, and the servers to reconnect with each other and bring the EMR application back online.

Most outages averaged 1-2 hours and required the team of 4-6 developers (\$50-80/hr) to search each server for that one service/process.

Implementing monitoring alone saved the company no less than \$200 per outage, just for the development team's time, let alone the cost of the entire hospital staff whose work was held up.

Also a factor was that reducing the downtime from one hour to 30 minutes meant we only missed/delayed one appointment instead of two or three. The impact on delayed surgeries was even more noticeable.”

THE RX IS RESTARTING SERVICES

Shared by Cahunt

“The Rx Group agreed to monitor the few specific services that were critical and which caused the robots that dispensed medication to shut down.

With simple notification, outages went from an average of one to one and a half hours to 20-25 minutes and translated to a savings of \$266 in staff time per event.

On the business side, not having an extended delay in the robots dispersing medications was (literally) invaluable to the medical staff and patients when you consider the risk to health and possible lawsuits.”

THE RIGHT INFORMATION AT THE RIGHT TIME

Shared by Paul Guido, Systems Team, Regional Bank in South Texas

“We monitor approximately 100 data circuits. According to our records, a circuit goes out about fifty times a year due to carrier issues, floods, dry spells, drunk drivers, ice storms and – everybody’s everybody’s favorite – backhoes.

Before monitoring, we would have to wait from 30 to 120 minutes before IT knew about the issue. To see if the issue was on our side, a person would be dispatched to the site. Typical drive time would be an hour. If the smart jack shows an error on the carrier side, a ticket would then be opened manually.

In addition, circuit outages cost a branch operation six to eight additional hours per circuit incident.

After implementing monitoring automation, a ticket is opened with the proper carrier within ten minutes. Our company and the carrier investigate the outage to see what is at fault (us or them). Because all the required information is gathered and included in the ticket, every hour of down time only causes three to four hours lost at the branch.

In a single year, circuit monitoring saves 250 to 400 hours of productivity.”

THE PRICE OF A CUP OF COFFEE

Shared by Kimberly, SysAdmin

“We have an application that approximately 470 developers use to drive their development cycle. The application would occasionally hang during a re-indexing overnight, and wouldn’t be

available when the majority of the devs arrived at work between 7 and 8 a.m.

I'm more of a 9 o'clocker myself, and so I'd either get a call during my not-so-awake moments, or they would have to wait for me to arrive at the office. Our devs make about \$33.50/hour.

Prior to automation, the math looked like this: 470 developers down for two hours at \$15,755 per hour, or approximately \$31,508 in lost employee productivity.

I was able to set up a monitor for that re-indexing job, and when it hung, executed a script that restarted the service and sent a message to our operations center to look at the app, confirming that it was accessible after the restart.

From that point on, the devs could start to work right away, the company didn't lose employee productivity, and I got to drink my first cup of coffee uninterrupted. Truly priceless!"

THE LITTLE THINGS ADD UP

Shared by Jason Higgins - Network Analyst

We have an application/service on a print server that allows our billers to print remotely stored data locally through encrypted channels. This application/service would frequently (two to six times a week) hang up and stop responding on the server. The fix is simple enough, just remote into the server, find the service, and restart it, which took about 10 minutes total from call to resolution.

The problem comes in when people don't report the problem all day, and then call in right at the end of the day in a panic. Our help desk consists of only three people, and they are frequently not at their desks because they are out working on problems. The lead person for the week carries a side phone to take calls on the go, but even then, you have to find a computer to get on, and so on and so forth.

I set up a job to monitor that service and alert when it was not running or not responding. This was a great first step because they help desk could see the alert, remote into the server, and restart the service. Usually this took place before the end-users even knew it wasn't working.

But again, this went on anywhere between two and six times a week. After having the alert set up for a few weeks, I discovered I could go a step further and actually restart the service when it stopped responding. I put this into place, and now the help desk does not have to do anything when the error happens. It takes care of itself, and they just log the ticket.

We calculated the cost of a single outage. In terms of staff cost, it was just \$16.31. But this is one of those cases where the math adds up. Two to six incidents per week means \$32-97 per week, which adds up to \$1,696 to \$5,044 per year. Maybe not a big deal to some larger companies,

The Completely Unnecessary Summary

but it was certainly a big deal to us.”

With all of this said and done, sophisticated alerts are enabled by (and are the result of) good monitoring. When done correctly it’s elegant, simple, and most importantly, not artisanal. It’s just monitoring and alerting the way it was always meant to work.

There are certainly many other examples of sophisticated monitors and alerts than the ones provided in this guide, but what I want you to leave with is the understanding that the biggest barrier to implementation at most companies is not the wrong tools, or the wrong skills. It’s having the wrong mindset, one that says monitoring and alerting is complicated or difficult. “Far beyond the ken of mortal man,” to quote the old Superman reruns¹.

In the end, monitoring and alerting are only limited by your ability to imagine and then implement, assuming you’ve got a good monitoring tool in place, rather than your ability to perform some weird interpretive dance.

¹ Warner Bros. and DC Comics, Inc., Superman, 1952.

Appendix A: What Can Go Wrong With IIS?

First, a little background on application pools:

Introduced in IIS version 6.0, application pools are used by IIS to isolate web applications. This allows you to have different configurations (security, resource usage, etc.) to help prevent misbehaving applications from interfering with other applications.

It is difficult to overstate the importance of this concept as it relates to an IIS-based web environment. The application pool is effectively the heart of a website. A pool isolates a web worker process, which brokers all communication between the URLs that make up a website and the kernel-level http.sys process that runs for the entire server.

Associating different URLs with their own application pool allows the server to use that single http.sys process to interact with different sites running on the server in different ways, and for each of those sites to be segmented from each other in terms of performance and security.

Conversely, associating multiple web sites with the same application pool allows them to run within the same security and performance context, and share information in a way that two separate web sites could not.

SO... what can go wrong with IIS?



To answer this question, it helps to explain what can't go wrong anymore. By separating the kernel-level http.sys process from the higher level web interface activities, a hang or crash at the web site level doesn't automatically hang or crash the web server.

What this means is that, prior to IIS 6.0, if you were browsing a web site that had a hang or crash, you would know relatively immediately. But now a failure in one website will not affect your browsing session on another site. If it can recover fast enough, it may not even impact your browsing session on the site with the problem.

CRASH VS HANG

Practically speaking, a crash is something that will wipe out the host process and stop whatever server side work and response generation that the process was supposed to send out. By using application pools http.sys holds the connections in kernel mode, so the connection stays connected and IIS starts up a new process to handle future requests.

On the other hand, a hang will usually keep its host process around, but no work will actually be done.

It's important to note that from the user's perspective, a crash or a hang are identical experiences that are indistinguishable from a browser error on their local system.

Dedications

TO DEBBIE

Mr. Rogers once gave viewers 10 seconds to think about the "...special ones who have loved us into being ... people who have helped you become who you are." You are the special one who has loved me into being, who has been by my side encouraging me to be the best version of myself. If I have achieved anything of real significance in this world, it is only because of you.

TO THE SOLARWINDS HEAD GEEKS

You are four of the most incredible, talented, intelligent, and exciting people to be around. I am grateful every day that I get to be part of this team, and do the work we do. Thank you for letting me be in the cool kids' club.

TO THE SOLARWINDS THWACK MVPS

Many of your words and wisdom appear on the pages of this book, either directly – because you are so generous with your time and skills – or as a result of our conversations in the convention booth, at SWUG™ meetups, on Slack®, and of course on THWACK.com itself. You inspire me with your creativity and enthusiasm. When I am representing SolarWinds, part of my mission is to also represent your importance and impact, to the company, to our products, and to me.