solarwinds

MONITORING 101

A primer to the philosophy, theory, and fundamental concepts involved in systems monitoring

by Leon Adato

ABOUT THIS GUIDE

This guide was written to provide an introduction to monitoring for someone who is familiar with computers and IT in general, but not with monitoring as a discipline. As such, (almost) no former knowledge or experience is required before delving into this document.

If you already have experience with monitoring, this may not be the book for you. Then again, it couldn't hurt.

This guide is completely tool-agnostic. While the author has decades of experience with a wide variety of monitoring solutions, those experiences were used only in the sense that provided insight into general trends and techniques. Nothing in this document will prepare you to actually dive into monitoring using software package XYZ.

But, it is my hope that after reading this, when you do start using monitoring software XYZ, some of the functions will be less arcane, and you'll be able to say *"Oh, I know why they are asking me for that option."*

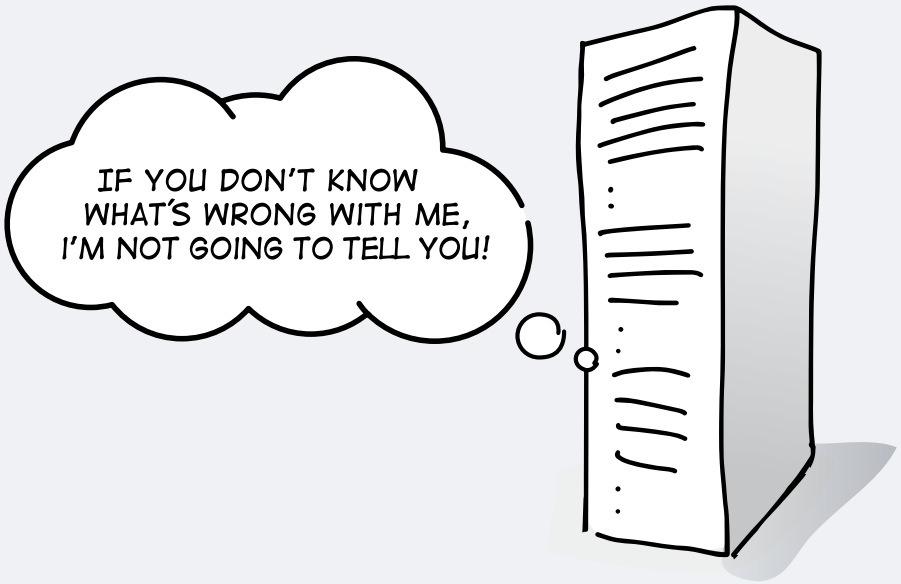


TABLE OF CONTENTS

IT WAS GOING TO BE A GREAT MORNING

(or "Why do we need monitoring?")

You get to your desk at 9 am sharp, having had a great morning workout, followed by a shower, a fantastic cup of coffee, and a frustration-free drive to the office. You are fresh and focused and ready to make a serious dent in that growing to-do list, which includes curious items like users complain that 'the Internet' gets really slow every so often, and the CFO thinks we're overpaying for WAN bandwidth. How much are we using?

Logging on to your PC, you notice that no emails have come in overnight. "That's odd," you're thinking.

Seeing you arrive, your buddy now walks over and says "looks like something's wrong with email."

You log on to the email server and find out that it's... well, you don't actually log on to the email server. Remote Desktop won't make a connection. You try pinging the box, and there's no response.

With a sinking feeling, you make the long journey to the computer room.

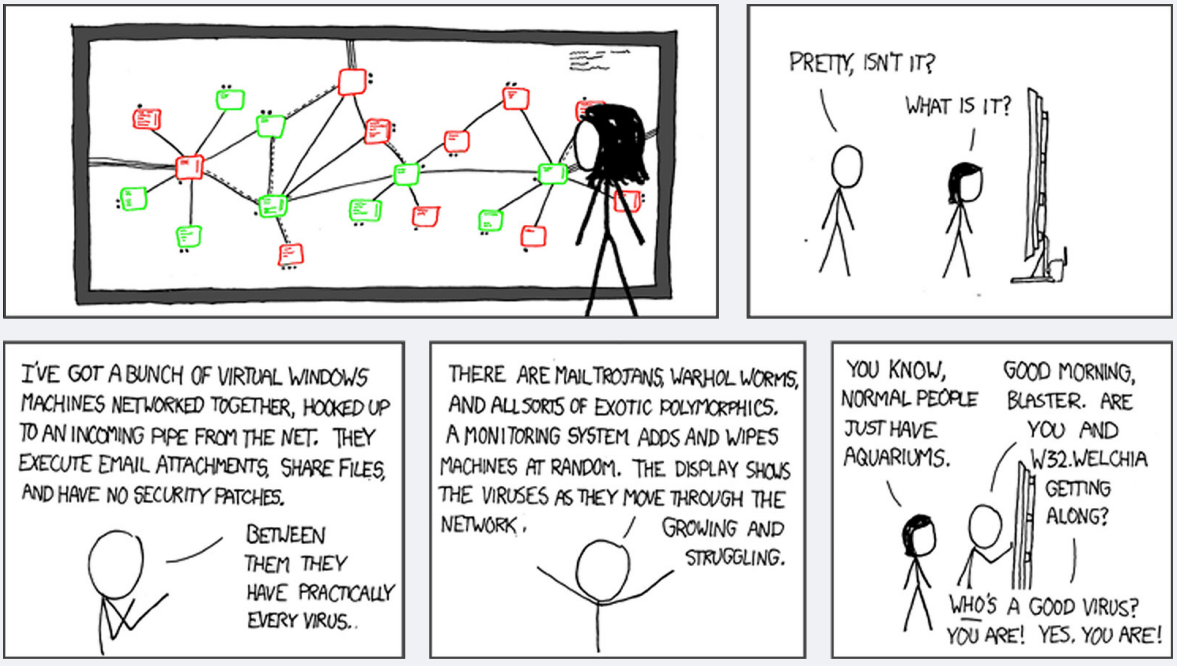
All hope of working on your to-do list is now gone as you stab a finger at the server's power switch. A few moments later you are logged on at the console. A pop-up alert on the screen tells you that one of the drives is completely full.

Much... (much!) later in the day, a picture forms of what happened. Sometime during the night (2:30 am to be exact) the data drive filled up, causing mail services to stop. Shortly after that, errors on the system drive reached a critical point, and the entire system crashed.

(Meanwhile, in the heat of fighting this fire, you didn't dig deeper to note that the data drive has been hovering at 95% capacity for over a week. And the drive that contains the operating system has been throwing read/write errors every 15 minutes for the last 17 days.)

About this time, your manager, who's been keeping a respectful distance while you worked, lets you know that the CEO is back from his contract discussions overseas. During the flight home, the CEO needed to send some follow-up documentation to the customer. When the corporate email wasn't responding, he resorted to creating a professional-sounding Gmail account and sent the files from there. The three of you are scheduled to sit down and debrief the situation in 30 minutes.

You start to pull some notes together for what you predict will be an uncomfortable conversation...



http://xkcd.com/350/

INTRODUCTION

If you have worked in IT for more than 15 minutes, the situation described above will strike you as neither unique nor rare—even if it is somewhat colorful. Systems crash unexpectedly, users make bizarre claims about how "the Internet" is slow, and managers request statistics, which leaves you scratching your head wondering how to collect them in a way that is meaningful and doesn't consign you to the hell of hitting Refresh and spending half the day writing down numbers on a piece of scratch paper just to get a baseline for a report.

The answer to all these challenges (and many, many more) lies in effectively monitoring your environment, collecting statistics, and/or checking for error conditions so that you can act or report effectively when needed.

Of course, this is easier said than done. Saying "let's monitor our network" presumes that you know what you should be looking for, how to find it, and how to get it without impacting the system you are monitoring. You're also expected to know where to store the values, what thresholds indicate a problem situation, and how to let people know about a problem in a timely fashion.

Yes, having the right tool for the job is more than half the battle. But, it's not the whole battle, and it's not even where the skirmish started.

To build an effective monitoring solution, the true starting point is learning the underlying concepts. You have to know what monitoring is before you can set up what monitoring does.

This document is designed to introduce you to the underpinnings of monitoring techniques, theory, and philosophy, as well as the ways in which monitoring is accomplished.

At no time will there be a discussion of specific software. At the end of the day, ping is still just ping, no matter how pretty a wrapper you put around it.

MONITORING CONCEPTUALIZED: THE FCAPS MODEL

Sometimes it helps to structure a concept into a model or imaginary framework, so you can fit all the new concepts into an overarching structure. Luckily, monitoring already has one of these, called FCAPS.

FCAPS stands for Fault, Capacity, Administration, Performance, and Security. Using the image of an airplane flying from one city to another, the FCAPS model would look something like this:

FCAPS	What It Tells You	On the Airplane
Fault	What's up?	Is the plane in the air, at the gate, or has it crashed?
Capacity	How much?	How many people are on the plane? How many seats are unoccupied? How much fuel is in the tank?
Administration	Who could?	Who purchased tickets for this flight, and, therefore, is allowed on the plane (whether or not they are on right now)?
Performance	How fast?	How fast is the plane going? Is it using the optimal fuel-oxygen mix? How many miles per gallon is the plane getting?
Security	Who did?	Who got on the plane?

While this is certainly a simplification of what can be a very broad subject, most of it holds true most of the time.

Monitoring is largely concerned with the F, C, and P of FCAPS. Administration (who has access to a system) and Security (who actually accessed the system at any particular time) is usually the purview of the security team and/or your RADIUS/TACACS-type tools.

Thus, the rest of this document will focus on F, C, and P, and leave A and S to go outside and play in traffic.

BUILDING BLOCKS OF A MONITOR

Regardless of the software you use, or the protocol, or the technique, there are a few fundamental aspects of a monitoring system that exist across the board. These are the basic components of a monitor. Not every building block will be present in every software tool or specific monitoring technique, but they are concepts you should understand, and you should realize that they all could factor into your work.

Element

This is a single aspect of the device you are monitoring, which returns one or more pieces of information. An example of this is an IP address, which could yield three data points: 1) whether it is responding, 2) how quickly it is responding, and 3) any packets dropped during the response.

Acquisition

How you get this information is another key concept. Does your monitoring routine wait for the device to send you a status update (push) or does it pro-actively go out and poll the device (pull)? What protocol does it use to connect to the target device?

Frequency

Closely tied to acquisition is how often information comes back. Does the device send a "heartbeat" every few minutes? Does it send only data when there is a problem? Does your polling process go out every five minutes and collect the element statistics?

Data retention

Monitoring, by its very nature, is data-intensive. Whether the acquisition method is push or pull, those statistics typically have to go somewhere and they pile up pretty quickly. At its simplest level, data retention is a Yes or No option. Either the statistic is 1) collected, evaluated, acted upon, and then forgotten, or 2) data is kept in a datastore. But looking deeper, data could be kept in a rolling log, which is a certain size, or spans a particular amount of time, and then the data "rolls off" or is deleted from the log. Or, it could be kept in a log which archives at a predetermined interval (when the file grows to a certain size, when a particular time period has elapsed, etc.), or data could be kept in a more robust datastore (i.e. a database).

Data aggregation

After you keep statistics for a while, it might become clear that the granularity of data isn't needed after a particular time. Aggregation is the act of consolidating, or averaging, a range of data points into a single number. For example, you might collect statistics every five minutes. After a week, those five-minute values are aggregated to an hourly average (thus 12 records crunch down to one). After a month, those hourly values are further aggregated to a daily average. In this way, database usage is optimized for short-term detailed analysis with a fuzzier long-term historical view.

Threshold

Referring back to FCAPS, the idea of fault monitoring is that you collect a statistic and see if it has crossed a line of some kind. It can be a simple line (is the server on or off?) or it can be more complex. Regardless, that line, which is crossed, is called a threshold. Examples of thresholds include:

- **SINGLE TRIGGER**

  This is where a single condition has been breached. Single condition does not necessarily mean a simple when x is greater than 50 formula. The condition could be complex (i.e. when the current time is after 8:00 am and before 5:00 pm, and x is greater than 50, and y is less than 20). But despite the possible complexity of the trigger itself, it is a single line in the sand.

- **DELTA**

  This threshold doesn't look at a fixed point; it looks at the rate of change. For example, when disk utilization goes up by 20%.

- **OCCURRENCE COUNT**

  Typically, in combination with a single trigger, this measures the number of times something occurs. For example, when %CPU goes higher than 85%, and it happens 5 times in a row over 15 minutes.

Reset

Reset is the logical opposite of threshold. It marks the point where a device is considered "back to normal." In a simple example, if a threshold is "when the device

is down," the reset would be "when the device is up." But, it's not always an exact match. A threshold might be "when disk utilization is over 85%," but the reset could be "when disk utilization goes below 70%." Like thresholds, a reset could occur based on a single trigger, a delta, or even an occurrence count.

Response

With all the discussion about thresholds and resets, the next logical question might be "OK, so then what?" What happens when a threshold is breached? Response defines that aspect. A response could be to send an email, play a sound file, or run a predefined script. The potential options are limited only by the capabilities built into the specific monitoring application. But the concept remains the same.

Requester (local agent or remote system)

With all the talk about monitoring, little has been said (yet) about where the monitoring is occurring, meaning, from what point in the environment are the monitoring statistics being requested (or to what location are they being sent, if we're talking about a push)? In its simplest terms, you have two choices: 1) a piece of software running on the monitored device itself (i.e. an agent), or 2) some location outside of the monitored device (agentless). More (a whole lot more) will be discussed on this concept later in this guide.

Authentication

Tightly tied up with the requester is authentication. How does the requester (whether agent or agentless) have the permission to ask for or receive the monitoring statistics? Authentication methods vary wildly, and are usually specific to particular monitoring techniques. We'll go into more detail about this later. But for now, understand that the part and parcel with the ability to monitor something is the permission to monitor it.

MONITORING TECHNIQUES

After all the theory and concepts, it's time to get down to some of the nitty-gritty. In this section, we'll discuss the various techniques that can be used to monitor.

It's important to note that, with possibly a few esoteric exceptions, this list comprises all the ways monitoring data can be collected. That fact can get lost when working with actual monitoring software. Many vendors of monitoring solutions like to appear as if what they are doing is super-ninja-voodoo-magic, and that nobody does it like them.

This is (to put it politely) a pile of horse feathers.

The software vendors are still using the techniques below. Where the sophistication comes in is with the frequency, aggregation, the relevance of displays, the ease of implementation, and other aspects of "packaging." More on that later.

One final note: the list below is a brief overview of each technique. It is by no means an in-depth analysis. For that, there is always Google.

PING

Good ol' ping. That great granddaddy of all monitoring. Ping sends out a packet to the target device, which (if it is up and running) sends an "I'm here" type response. The result of a ping tells you whether or not the device is responding at all (i.e. up), and how fast it responded.

Usually, ping monitoring sends out multiple packets, and calculates up/down based on how many positive responses come back. Some tools send out just one ping. Some send one ping, and if it doesn't come back, send out more to verify whether the device is really down or just sluggish.

SNMP

SNMP (Simple Network Management Protocol) has a few pieces that combine to provide a powerful monitoring solution.

First, SNMP is comprised of a list of elements that return data on a particular device. It could be CPU, the average bits per second transmitted in the last five minutes, or the number of errored frames. Each of these elements is assigned a numeric ID

(object ID or OID), which are grouped hierarchically. The entire collection of OIDs is contained in a Management Information Base (MIB).

Second, an SNMP service (also called an agent) runs on the target device, keeping track of each of those elements and what their current values are. While it seems like a lot of data, the agent is running locally, and the data points are small and easily managed. Plus, the agent is only keeping the current number, so there is no on-going storage of information.

Finally, that agent provides that data based on either an SNMP Trap trigger, or an SNMP Poll request. The combination of both allows you to collect information about a device, either at preset collection intervals (polls), or only when a specific threshold has been exceeded (traps).

SNMP Trap

A trap occurs when one of the internal data points crosses a threshold (that threshold is also indicated within the MIB). When that trigger is crossed, the SNMP process running on the target device will pro-actively send out information about the OID that was triggered. The SNMP agent is configured to send that information to another system in your environment, called a Trap destination.

SNMP Poll

An external device can also request the data for an OID at any time by polling the device. This is also called an SNMP GET request.

SYSLOG

Syslog messages are similar to SNMP traps. A syslog service (again, agent) takes events that occur on the device and sends them to a remote listening system (Syslog destination server). Syslog differs from SNMP Traps in that syslog messages are relatively free-form, and aren't dependent on the MIB-OID structure that SNMP requires.

Syslog can be used as a replacement for traditional log file monitoring. Instead of sending a message to a log, it is sent via the syslog protocol to an external device—typically, one that collects syslog from multiple devices, and then acts on those triggers.

Syslog is often the monitoring option of choice for security devices such as firewalls and IPS systems.

In addition to being more free-form, Syslog tends to be "chattier" than SNMP Traps. However, it's more flexible because many applications can be set up to send syslog messages, whereas few can send out customized trap items without a lot of pre-work on the part of the developer.

LOG FILE

Log file monitoring is pretty straightforward. An application or process writes messages to a plain text file on the device. The monitoring piece of that comes in the form of something that reads the file and looks for trigger phrases or words.

For example, ERROR. Unless, of course, it's in a line that says NO ERROR FOUND.

EVENT LOG

Event log monitoring is specific to Windows®. By default, most messages about system, security, and (standard Windows) applications events are written here. When an event is written, the standard data elements are:

- Message: Event message controlled by the application.
- Category: Used to direct where the event will be logged.
- Priority: Used to determine if the event will be logged.
- Event ID: Used to classify events.
- Severity: Used to determine the event notification.
- Title: Further identifies the event.

Event log monitors watch the Windows event log for some combination of EventID, category, etc., and perform an action when a match is found.

PERFORMANCE MONITOR COUNTERS

Performance Monitor (or PerfMon) counters are another Windows-specific monitoring option, and can reveal a great deal of information, both about errors on a system, and ongoing performance statistics. Perfmon counters have the following attributes:

- The name of the machine.
- The category of the counter (e.g. processor).
- The name of the counter (e.g. % processor time).
- The instance (if any) for the counter. (e.g._total).

Perfmon counters can be collected remotely by a machine that has privileges on the server to be monitored. In addition to the hardware and operating system information that is present by default, many applications come with their own perfmon counters as well.

WMI

WMI (Windows Management Instrumentation) is, at its heart, a scripting language built into the Windows operating system. A wide variety of tasks can be performed via WMI, but for the purpose of monitoring, the focus is on collecting and reporting information about the target system.

The benefit of WMI is that, like perfmon counters, a remote system with the right privileges can run WMI scripts without having to be "on the box" using some sort of local agent.

Where WMI exceeds perfmon counters is in its ability to interact with the target device. Via WMI, you could run a script to read a directory, get the number of connected disks, or find out who the logged-in users are.

SQL QUERY

Just as the name suggests, this monitoring technique involves running a query against a database. The point of this query is to find out information about the database server itself. Thus, you would (typically) query the system tables to find out about the number of client connections, transactions per second, size of the transaction log, the number of active connections, etc.

SCRIPT

This is one of the easiest monitoring methods to understand, and is certainly the broadest in scope. Running a script to collect information can be as simple or complicated as the author chooses to make it. In addition, the script might be run

locally by an agent on the same device, and report the result to an external system. Or, it might run remotely with elevated privileges. All of this (and more) has to be determined by the script author and the monitoring specialist.

LOG FILE AGGREGATION

Log file aggregation, as mentioned previously, is the process of collecting log files (including the Windows Event Log) from multiple systems and bringing them to a central location. Once this is done, all the information from those logs is combined in a way that lets the aggregation system track which device the log originated on.

IP SLA

IP SLA (short for Internet Protocol Service Level Agreements, which explains nothing about what this does) is a pretty comprehensive set of capabilities built into Cisco equipment (and others nowadays, as they jump on the bandwagon). These capabilities are all focused on ensuring the voice-over-IP (VoIP) environment is healthy.

Functionally speaking, IP SLA allows you to set up certain actions to occur on a network device, and have the results of that operation reported back to a remote server. Examples of the types of operations include:

- Check if a webpage is responding.
- Check if a DNS server is responding and resolving names correctly.
- Check if a DHCP server is responding and handing out IP addresses (and that it has addresses available to hand out).
- Set up a fake phone call with another IP SLA-capable device, and collect statistics about the call quality including jitter, packet loss, MOSS and more.

This is a big deal because it uses the devices that are part of the network infrastructure, rather than requiring you to set up separate devices (or agents on existing PC's or servers) to run tests. In addition to being able to test performance between specific points on the network, certain tests (like DHCP) can only be done on the same network segment where the responding device is located.

NETFLOW

Standard monitoring can tell you that the WAN interface on your router is passing 1.4Mbps of traffic. But who is using that traffic? What kind of data is being passed? Is it all HTTP, FTP, or something else?

NetFlow answers those questions. It sets up the information in terms of "conversations"—loosely defined as one session of data transfer between two computers that use the same protocol. So if DesktopComputer_123 is sending a file to Server_ABC via FTP, that is one conversation. If the same PC is also browsing a webpage on the same server using HTTP, that is a different conversation. If the same PC is also streaming a video from YouTube (also HTTP), you have a third conversation.

NetFlow data is captured by a network device that is somewhere in the middle of the conversation—usually one or more routers near the center of the network (or one router at each remote location if there are site-to-site communications that don't go through the core). This device collects the NetFlow data and sends it to the monitoring server every so often. It is up to the monitoring server to aggregate, parse, and analyze the data.

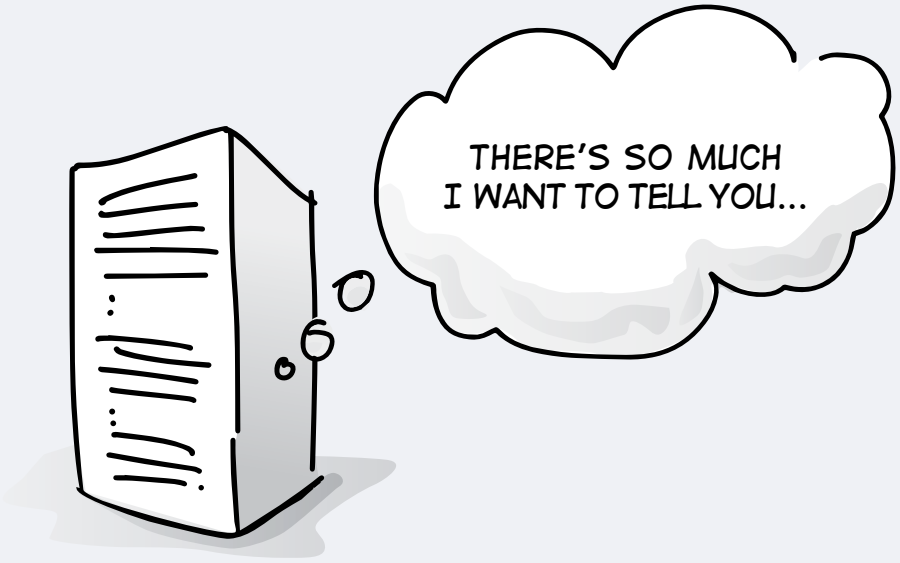
Note that the two machines in the conversation (DesktopComputer_123 and Server_ABC, in my example) do NOT need to be monitored. Just the core device. Conversely, the core device must be capable of capturing and sending NetFlow data. Many (but not all) Cisco routing devices can do this. Other brands/models might also be able to do so, but you should check.

API

Using an API is the murkiest of all monitoring techniques. The essential concept to understand is that there are hooks built into an application that allow for a data request to be heard, and that data is sent as a response. This is all done in a way that is not already laid out in this document.

Two notable applications in this category (at the time of this writing) are Microsoft® and VMware®. The difference is that the Microsoft API is only available to its own MS SCOM monitoring product, while the VMware API is published, and several vendors take advantage of it.

The upshot is that if your monitoring software can take advantage of this, that's great. If not, there is not much you are going to do about it short of writing your own program that requests and handles data from this source.



TIPS, TRICKS, AND TRAPS

This section lists out various monitoring-related issues, gotchas, tricks, or philosophical ideas that may help you get a leg up on your work. They are listed in no particular order.

FLAPPING (OR "SAWTOOTHING")

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.

Here are a few techniques that can be used to avoid this, depending on the toolset:

- Suppress events within a window. Some software can ignore duplicate events during a period of time.
- Add a delay before triggering. If you need to ensure that a failure condition is absolutely real, some monitoring tools allow you to wait xx minutes (or for xx occurrences of the alert) before triggering an alert.
- Do not cut a new alert until the original has reset. More 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 its < 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 2 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.

PARENT-CHILD

Imagine a simple environment where you have a remote site with a router, a switch connected to the router, and a server connected to the switch. And, you

are monitoring from the home office. Assume you want to get alerts when a device goes down (i.e. ping stops working).

Without any special techniques, if the router goes down you are going to get (at least) three alerts—for the router, the switch, and the server.

Parent-child relationships (which typically have to be set up manually for each set of devices) are a way of telling the monitoring system what is connected and how. This way, when a parent device is down (the router is the parent of the switch, the switch is the parent of the server), any alerts related to the child devices are suppressed. When the router is down, the switch isn't (necessarily) down. It may be simply unreachable.

In more sophisticated tools, the software might actively check the upstream parent before marking a device as down, and continue all the way up the chain until it finds the highest level device that is down. This is known as upstream verification (or conversely, downstream suppression). In many cases, parent-child management is accomplished by adding an extra polling cycle before marking devices down IF they have a parent device. This way the parent device status can be verified in the normal polling routine, and doesn't increase the load on the server.

MONITORING A DELTA

In some cases, such as when monitoring disk space, you may not be interested in a specific numeric threshold (alert when the disk is more than 90% utilized). This is because the disk sizes are so vast that even 1% is not enough to cause immediate alarm (remember, 1% remaining on a 1 terabyte drive is still 100+ gigabytes).

Instead, in some cases, what you want to monitor is the delta, or the rate of change. Here, you might be interested in knowing that disk utilization has gone up by more than xx% over yy minutes, which might indicate a spike in consumption. If left unattended, this could result in a disk becoming full within a short time.

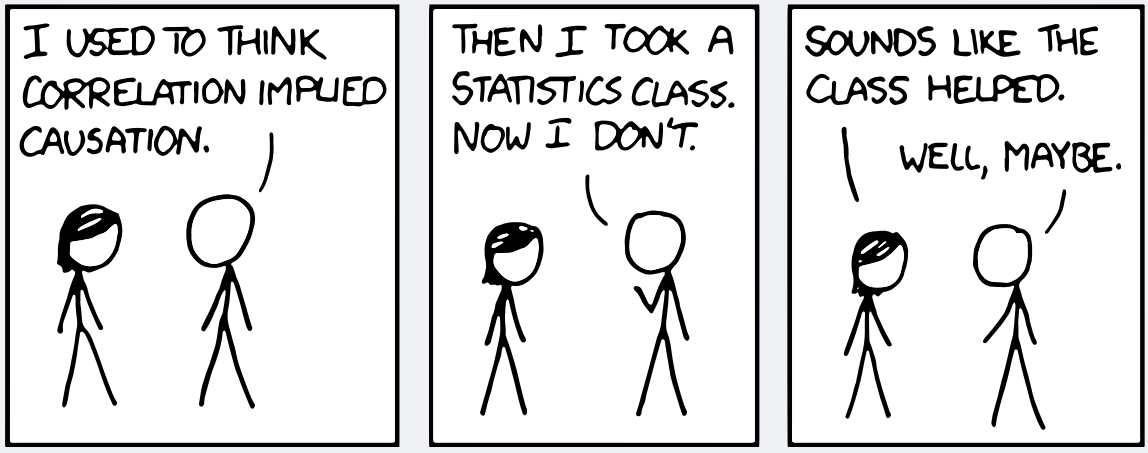
Certainly, the actual disk capacity is also a factor. But with many enterprise environments today, that threshold is very high.

EVENT CORRELATION

Event correlation is a big topic. It's much bigger than one section of this document can accommodate, but it's important enough to merit a brief discussion.

Event correlation tools can perform the flap detection and suppression, as well as parent-child correlation (described earlier). In addition, event correlation tools might perform the following:

- **On event X, look for Y:** When a single event comes in, search the recent events (in seconds or minutes) for a different event, and trigger if both have occurred.
- **On event X, wait YY minutes, and look for event Z:** Similar to the previous situation, but in this case, X is the causal or initial event, which starts a timer. If a second specific event is seen within the timer window, an alert is triggered.
- **De-duplication:** One of two techniques that "fix" flapping. De-duplication usually involves creating an alert on the first occurrence, and then ignoring all subsequent occurrences, unless 1) the original event created a trackable ticket, and that ticket was closed by a human operator, or 2) a certain amount of time has elapsed.
- **Alert after XX times:** The second of two techniques that address flapping. In this case, the flapping is itself an indicator. If a problem occurs once, it is negligible, but repeated occurrences indicate the presence of a problem. Therefore, the event correlation system needs to keep track of how many times a particular error occurs for a particular system. An alert is created when the count reaches a sufficiently high level (possibly within a specific time frame).



http://xkcd.com/552/

AGENT-BASED OR AGENTLESS?

One of the larger variations on monitoring techniques comes down to whether a tool works based on agents (small piece of software that runs on every device that needs to be monitored) or agentless. Each has its benefits and drawbacks. Very few monitoring solutions use a truly hybrid approach. Most emphasize one over the other, significantly, if not completely.

Agent-based has the benefit of not being network-dependent (the agent runs as long as the device is running and stores data locally until it can be forwarded to a listening server). It usually runs with elevated privileges, enjoys fast execution times (running as fast as the device itself can run), and is able to run certain actions that simply do not work between two devices across a network connection.

The downside of agent-based monitoring is that a great deal of effort needs to be put into deploying, maintaining, and monitoring the health of the agent itself. In addition, there are times when a monitoring agent can actually cause problems on a device rather than remaining out of the way and monitoring conditions.

Agentless, on the other hand, is generally innocuous with regard to its impact on the device being monitored. Even when an agentless monitoring system launches a script that executes on the device to be monitored. It would be a poorly written script, rather than the monitoring tool itself, which could have an unforeseen negative impact on the performance of the device. And, because there are no agents to deploy, monitoring is often a matter of maintaining a centralized database of devices that need to be polled on a regular basis. Adding more devices is simply a matter of expanding the list.

Of course, the trade-off is that agentless monitoring is often not able to accomplish as much with regard to the granularity of data collected as well as the robustness of an automated response in the event of a problem. In addition, because all polling occurs from a centralized server, the load on the monitoring infrastructure is greater. Therefore, you often need more servers to accomplish what would be distributed to agents.

All of that said, the agent vs. agentless debate is one of selecting the right tool for a particular job—not one of “good” versus “bad.”

SO WHAT SHOULD I BUY?

After all is said and done, you still need to buy or build a software tool that does some of the things described in this document. Because all the software vendors out there selling solutions are working from the same basic playbook, what should you look for as a differentiating factor? What is it, exactly, that makes brand X so much better than brand Y?

The answer has as much to do with you and your organization as it does with how monitoring gets done.

Will your monitoring team be one person who is also your server team and network team and helpdesk team and database team? If so, you probably need a tool that sacrifices comprehensive options for simplicity and manageability. Does your organization need absolute flexibility so that the monitoring solution is the one-stop-shop for all your needs? You will pay more, and require more staff, but at the end of the day (or month, or more likely year) you will have a software suite that fits you like a glove.

With all of that said, the non-technical items you should consider include:

- Cost to purchase and install. This includes hardware requirements and the specific needs for your environment. Do you need a separate system to monitor devices in your firewall and/or remote sites? How many monitoring systems do you need for all the devices in your company? And so on.
- Ongoing maintenance cost (i.e. license costs in year two and beyond).
- Support requirements. How many people are needed to maintain the system? This is one of those questions that you should NEVER trust the vendor to answer. Talk to some other companies that are using the software.
- How much customization is needed? Again, talking to other companies is extremely useful here.

For technical requirements, I strongly recommend checking out my feature-function matrix, which allows you to compare the technical merits of various toolsets:

- ODS (open document) version: http://thwack.solarwinds.com/docs/DOC-170962
- XLS Version: http://thwack.solarwinds.com/docs/DOC-170959
- XLSX Version: http://thwack.solarwinds.com/docs/DOC-170961

After all is said and done, you still need to buy or build a software tool that does some of the things described in this document. Because all the software vendors out there selling solutions are working from the same basic playbook, what should you look for as a differentiating factor? What is it, exactly, that makes brand X so much better than brand Y?

Obviously, we at SolarWinds have a fondness for our own tools. You can visit http://www.solarwinds.com for more information.

However, The answer has as much to do with you and your organization as it does with how monitoring gets done.

DEDICATIONS

To Debbie.

As with anything I have ever done that mattered, this is dedicated to my best friend and my closest confidant. Even after 3 decades, you still mean more to me than everything else.