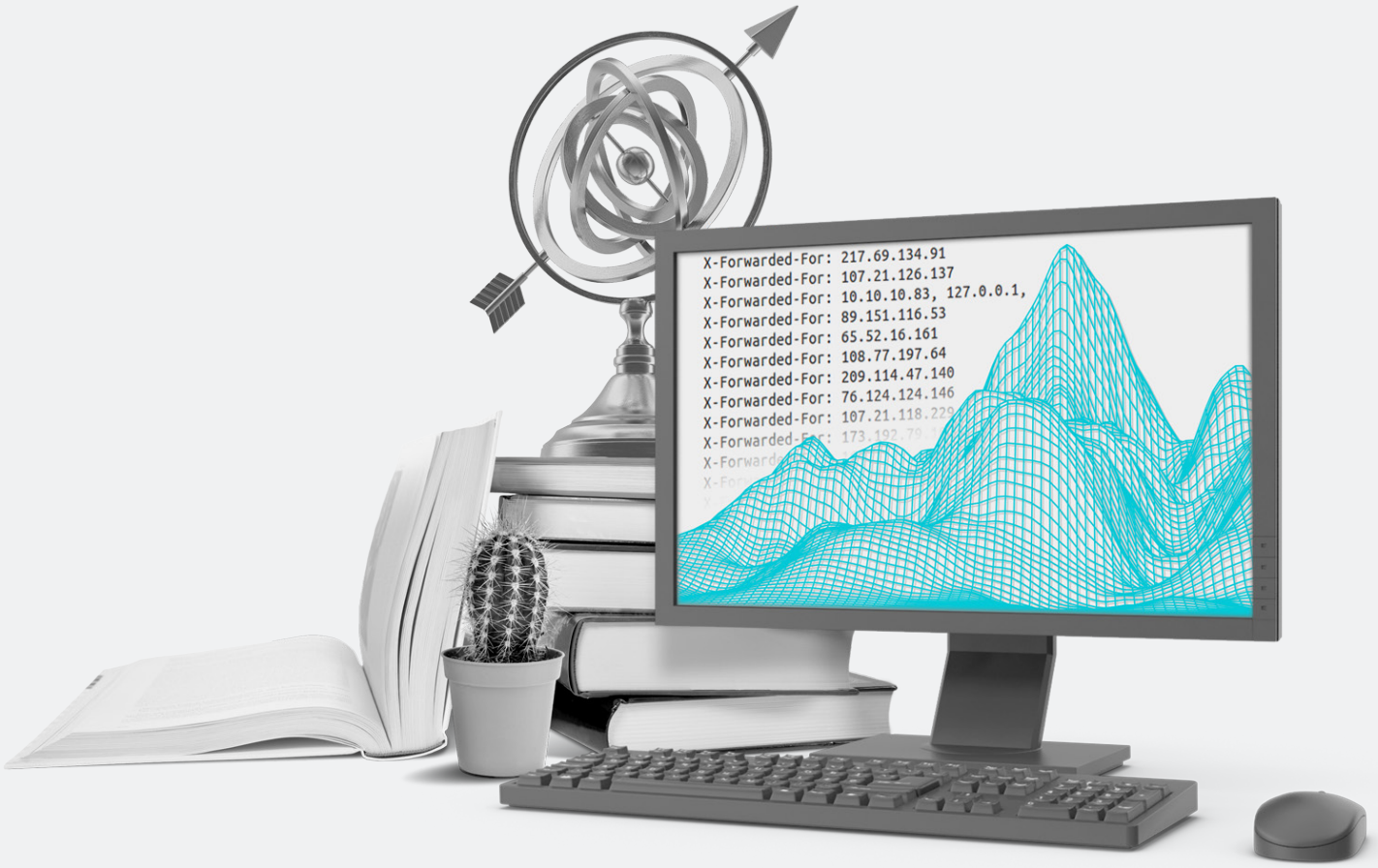
solarwinds



eBOOK

Monitoring 101:

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

by Leon Adato

About this Book

This guide was written to introduce monitoring to someone who's familiar with computers and IT in general, but not with monitoring as a discipline. As such, (almost) no former knowledge or experience is required to delve 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 only used to provide insight into general trends and techniques. Nothing in this document will prepare you specifically to dive into monitoring using software package XYZ.

But, I hope 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're asking me about that option."

About Version 2.0 of this Book

When I originally wrote this book back in 2012, the monitoring landscape—both in terms of what technologies were available and what people wanted to accomplish—looked markedly different than they do here in not-quite-2020. Back then, the word "observability" didn't carry the weight it does today. Logfile aggregation tools were mostly used for security teams and little else. While a few vendors had APIs, those were used for programmatic interaction, not data collection. And application tracing was almost exclusively used by the developers in the company to spot-check their code as they created a new application.

All this (and more) has changed today. I still stand by my assertion monitoring is, at its heart, simple (which isn't necessarily the same thing as "easy"). But the half-dozen protocols and techniques relied on by monitoring over the last decade has expanded a bit, and therefore I realized this book, if it was to continue to be useful and relevant, would have to keep pace with the times.

About the Author

In my sordid career, I've been an actor, bug exterminator and wild-animal remover (nothing crazy like pumas or wildebeests – just skunks and raccoons), electrician, carpenter, stage-combat instructor, American Sign Language interpreter, and Sunday school teacher.

Oh, and I work with computers.

Since 1989 (when a free copy of Windows 286 came on 12 5¼" floppies when you bought a copy of Excel 1.0), I've worked as a computer classroom instructor, courseware designer, desktop support tech, server administrator, network engineer, and software distribution expert.

Then, 20 years ago (give or take), I got involved with systems monitoring. I've worked with a wide range of tools and designed solutions for companies from the extremely modest (~10 systems), to the fairly large (5-10,000 systems), to the ludicrously enormous (250,000 systems in 5,000 locations). During that time, I've had to chance to learn about monitoring all types of systems—routers, switches, load balancers, and SAN fabric; applications running on Windows, Linux, and Unix; and servers running on physical, virtual, or cloud-based platforms.

About SolarWinds

(i.e. "Shameless Self-Promotion")

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

First, SolarWinds is Geekbuilt.® Meaning geeks, including SysAdmins, engineers, and other IT professionals, produce solutions for other geeks. SolarWinds addresses real problems geeks face every day at work. We don't design 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're thinking about in terms of problems, but also how they'd like to see those problems addressed. Their 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 snap together under a common framework, and also integrate well with solutions from other vendors. Because real geeks know you don't get to pick every single piece of software the company uses, and 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 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. Head over to solarwinds.com for more detailed information on products and pricing; or to download a free, unlimited (meaning you can load up as many devices as you want), 30-day demo of any most (or all) of the SolarWinds modules.

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

It Was Going to Be a Great Morning

(or "Why do we need monitoring?")

You get to your desk at 9 a.m. 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're fresh and focused and ready to make a serious dent in your growing to-do list, which includes curious items like users complaining about how "'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 no emails have come in overnight. "That's odd," you think.

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

You log on to the email server and discover 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're logged on at the console. A pop-up alert on the screen tells you 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 a.m. to be exact) the data drive filled up, causing mail services to stop. Shortly after, 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 the data drive has been hovering at 95% capacity for *over a week*. And the drive containing 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 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…

Introduction

If you've 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 meaningful way that 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 you can act or report effectively when needed.

Of course, this is easier said than done. Saying "let's monitor our network" presumes you know what you should be looking for, how to find it, and how to get it without affecting the system you're 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.

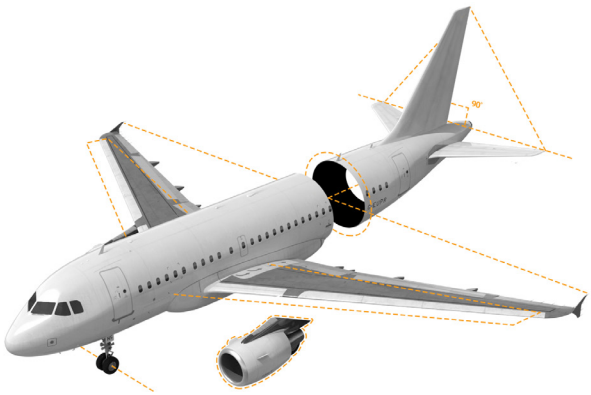
To build an effective monitoring solution, the true starting point is learning the underlying concepts. You have to know what monitoring *is* before you can set up what monitoring *does*.

This document is designed to introduce you to the underpinnings of monitoring techniques, theory, and philosophy, and 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, which allows you to 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, on the ground, 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're 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?

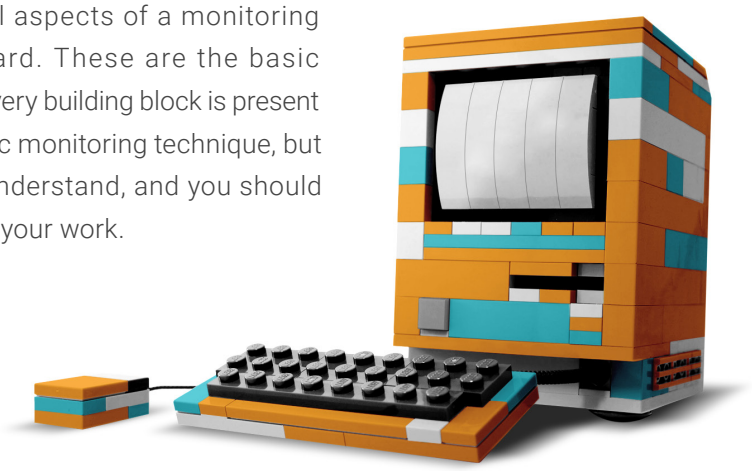
While this is certainly a simplification of what can be a broad subject, most of it holds true most of the time.

Monitoring is largely concerned with the F, C, and P of FCAPS. Administration (who has access to a system) and Security (who 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, a few fundamental aspects of a monitoring system exist across the board. These are the basic components of a monitor. Not every building block is present in every software tool or specific monitoring technique, but they're concepts you should understand, and you should realize they all could factor into your work.



Element

This is a single aspect of the device you're 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's responding, 2) how quickly it's 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 proactively 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's 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 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 specified 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 the granularity of data isn't needed after a 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 to collect a statistic and see if it's 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, the line crossed is called a threshold. Examples of thresholds include:

- **Single trigger**

  This is where a single condition has been breached. Single condition doesn't 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 a.m. and before 5:00 p.m., and X is greater than 50, and Y is less than 20). But despite the possible complexity of the trigger itself, it's 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 five 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, then what?" What happens when a threshold is breached? The 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 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 you can use to monitor.

It's important to note, with possibly a few esoteric exceptions, this list comprises all the ways monitoring data can be collected. This fact can get lost when working with actual monitoring software. Many vendors of monitoring solutions like to appear as if they're doing super-ninja-voodoo-magic, and 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's always Google.

PING

Good ol' ping. The great granddaddy of all monitoring. Ping sends out a packet to the target device, which (if it's up and running) sends an "I'm here"-type response. The result of a ping tells you whether the device is responding at all (i.e., up), and how fast it responded.

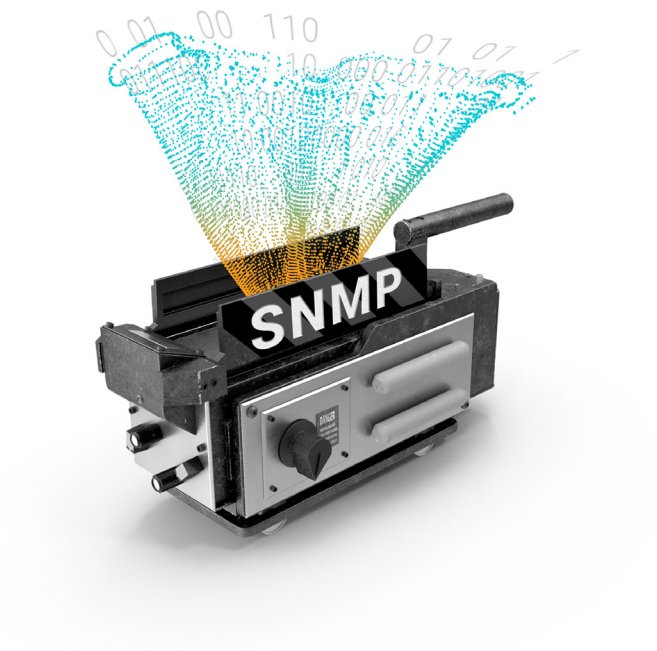
Usually, ping monitoring sends out multiple packets, and calculates up/down based on how many positive responses come back. Some tools send out just one ping. Some send one ping, and if no response comes back, send out more to verify whether the device is truly down or simply sluggish

SNMP

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

First, SNMP is comprised of a list of elements returning 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 their current values. 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's no ongoing storage of information.



Finally, the agent provides 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 the trigger is crossed, the SNMP process running on the target device will proactively send out information about the OID being triggered. The SNMP agent is configured to send the 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.

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.

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 with privileges on the server to be monitored. In addition to the hardware and operating system information present by default, many applications come with their own perfmon counters.

IP SLA

IP SLA (short for Internet Protocol Service Level Agreements, which explains nothing about what this does) is a 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 the 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 whether 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 PCs 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 the WAN interface on your router is passing 1.4Mbps of traffic. But who's using the 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 using the same protocol. If DesktopComputer_123 is sending a file to Server_ABC via FTP, this counts as one conversation. If the same PC is also browsing a webpage on the same server using HTTP, it's a different conversation. If the same PC is also streaming a video from YouTube (also HTTP), you have a third conversation.

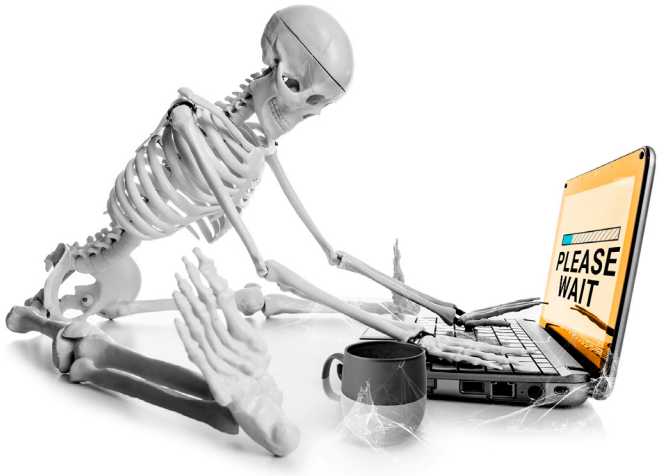
NetFlow data is captured by a network device located somewhere in the middle of the conversation—usually one or more routers near the center of the network (or one router at each remote location if there are site-to-site communications which don't go through the core). This device collects the NetFlow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

Note 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.

SQL QUERY

Believe it or not, the database itself has many secrets to give. Most DBAs are interested in "waits," "locking," and "blocking"—which represent the amount of time a user (or an application working on behalf of the user) can't do its job—getting data out or putting data in—because the database is busy doing something else for someone else.

As the name suggests, this monitoring technique involves running a query against a database to find information about how the database server itself is performing. Thus, you would (typically) query the system tables to find out about the waits, locking, and blocking mentioned earlier, along with things like the number of client connections, the transactions per second, or the size of the transaction log.



SYNTHETIC TRANSACTIONS

There's nothing more frustrating than discovering a system or application has been down for hours, but the outage was undetected because no users were on it to notice and report it. Synthetic transaction monitoring simulates the actions a user would make and repeats those actions on a regular basis. If the process fails or takes longer than expected/required, the monitoring system can create an alert and the issue can be addressed before actual users get on the system. This technique can also determine when the load on a system or application is so great it affects overall performance and user experience.

LOGGING

"You keep using that word. I do not think it means what you think it means"—Inigo Montoya, "The Princess Bride"

When people in IT say the word "logging," I honestly have no idea what they mean. I know what they *might* mean, but until I nail it down, it's a toss-up.

Here's a handy guide to identify what people are talking about. "Logfile monitoring" is usually applied to four different and mutually exclusive areas.

SYSLOG

Syslog is a protocol which allows one machine to send a message to a server listening on UDP port 514. This protocol is most often found when monitoring network and *nix (Unix, Linux) devices, although network and security devices such as firewalls and IDS/IPS systems send out their fair share as well.

Syslog messages are similar to SNMP traps, but differ in that syslog messages are relatively freeform and don't depend on the MIB-OID structure required by SNMP.

In addition to being more freeform, 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.

Syslog can be used as a replacement for traditional log file monitoring. Instead of sending a message to a local log file, they're sent via the syslog protocol to an external device—typically, one collecting syslog from multiple devices, and then acting on those triggers.

WINDOWS EVENT LOG

As the name strongly implies, Windows Event Logs are specific to the Microsoft Windows operating system. The event log isn't a log "file" at all, but a database unique to Windows machines.

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

LOGFILE AGGREGATION

Logfile aggregation 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 "normalized"—data elements are lined up so all the dates are in the same column and format, the source system is identified, and so on.

Combining the data in this way lets the aggregation system track on which device the log originated, but also allows users to set up rules to identify trends (such as failed logins) occurring across the enterprise, rather than on a single system; or anomalies— completely unique events across the environment which might be indicating an "unknown unknown" (more about that later).

MONITORING INDIVIDUAL TEXT FILES ON SPECIFIC SERVERS

This activity focuses on watching a specific (usually plain-text) file in a specific directory on a specific machine, looking for a string or pattern to appear. When a matching pattern is found, an alert is triggered.

Of course, it can get more involved than this:

- By monitoring a file with a name matching a specific pattern (like a date) instead of a specific fixed filename
- Or by monitoring the newest file in the newest directory inside a certain location
- Or by monitoring for a word or phrase (a "string") matching a particular pattern rather than a single fixed word or set of words
- Or by triggering when a pattern has occurred three times within a five-minute period instead of just once ever

Regardless of the complexity or variations, the goal is the same—to find a string or pattern within a file.

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.

API

I hesitated to include this (for reasons you'll see in a minute), but then I realized you would probably read about it in some marketing brochure, and I wanted you to have an idea of what it was talking about. I want to note that I'm explicitly not talking about APIs which use a JSON/REST framework. Those are 100% real and 100% useful, and I'll cover them in the next section.

When a vendor says their monitoring solution uses "an API" to collect data, this is the murkiest of all techniques or pseudo-techniques out there. Essentially, hooks are built into an application to allow for a data request to be heard, and data sent as a response. Theoretically, this is done using a technique not laid out anywhere else in this document.

If this all sounds kind of magically bull-crappish, you're not wrong. Very few solutions deliver on this miraculous promise. They're usually doing something relatively simple and then dressing it up to look super-neato-cool.

Two notable exceptions in this category are Microsoft and VMware®. The difference is the Microsoft API is only available to its own SCOM monitoring product, while the VMware API is published, and several monitoring vendors take advantage of it.

If your monitoring software can take advantage of this, great. If not, there's not much you're going to do about it short of writing your own program to request and handle data from this source.

JSON/REST

When I said "API" in the previous section, I explicitly left out JSON/REST because it's also often referred to as "API monitoring" and, in my opinion, it's the only API monitoring technique with which you should concern yourself.

But what's this technique about? How does it work?

Representational State Transfer (REST) refers to the method by which data is transmitted between systems—whether the data in question is unstructured text, XML, JSON, or something else. REST works by having a specific URL, which responds by sending a set of data instead of a regular HTML webpage. An example might be:

http://www.myapplication.com/api.php?id=MonitoringCompanies

If I go to this web page (whether via a browser or as part of a program), the server would return a set of data.

Meanwhile, Java Script Object Notation (JSON), which was developed in 2002, refers to the structure of the data. Its highlights include being "self-describing" (human-readable) and hierarchical (values within values), and it can be parsed and used by a wide range of programming languages.

An example of JSON data the URL above might returnwould look like this

```
{"Monitoring_Companies":[

 { "Name":"SolarWinds," "Location":"Austin," "Website":"www.solarwinds.com" },

 { "Name":"Pingdom," "Location":"Austin," "Website":"www.solarwinds.com/pingdom" },

 { "Name":"Papertrail," "Location":"Austin ," "Website":"www.solarwinds.com/papertrail" }

]}
```

Saying an application can be monitored via a JSON/REST API means a program is listening on the device and is prepared to deliver a set (or multiple sets) of data and statistics based on the parameters passed to it through the URL.

This opens up a significant amount of options for vendors who want to expose data about their application.

At the same time, because this is a (relatively) new method of monitoring and because the means by which data is collected is incredibly specific to each vendor's implementation, it presents a challenge to monitoring solutions to do, and in a standardized way.

APPLICATION TRACING

As I mentioned in the introduction to this book, once upon a time (meaning "five or six years ago"), application tracing was almost exclusively used by the developers in the company to spot-check their code as they created a new application. Now it's one of the pillars of observability (more on that later) and an important technique for monitoring insight across the enterprise.

The mechanism for tracing can vary, but the goal is the same—to understand the flow of actual transactions moving through the system, rather than the synthetic transactions described earlier. Tracing can uncover bottlenecks, whether they occur at the web interface, application, database, or post-processing phases. Robust solutions can get down to the individual line of code creating a slowdown or error, enabling rapid and precise troubleshooting.

Let's Talk About "Observability"

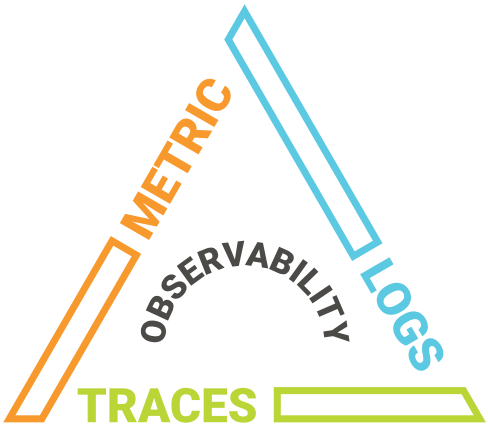
While a recently-favored buzzword in monitoring, the word "observability" has deeper roots and a larger implication for someone trying to wrap their arms (or their head) around monitoring as a discipline, and it deserves a section in this 101-level book.

Originally coined in the 1960s as part of the "control theory" branch of engineering and mathematics, a system is "observable" if the internal state can be determined from external outputs. A simple example would be an external display showing the internal temperature of an engine. For temperature, at least, the engine would be considered nominally "observable."

In true observability, the "engine" is a system (application, or more likely a set of interrelated applications) in a constant state of motion/processing. Observability means the state of every element of the application/set of applications can be known from the outputs. We don't have to stop the program and step through code, look at memory dumps, etc.

This idea took hold in the early 2010s, along with the rise of DevOps. In the race to create a culture able to own both the development and operational sides of IT; to rapidly deploy new changes into production (multiple times a day, if desired); to use monitoring not only to determine system performance and stability but also to extract lessons about user behavior and experience on the live system... to accomplish all those things, early leaders in this space realized they needed more than sampled metrics could provide. After all, if you're going to change an application on the fly and risk "testing in production," you need to know exactly what impact any given change was having, as close to real time as possible.

One tool or technique wasn't going to work for this, so the concept of "The Three Pillars of Observability"—metrics, logs, and traces—rose to prominence. But simply having monitoring tools to collect metrics, logs, and traces isn't enough. A solution providing true observability needs to be able to combine all the data for a new kind of insight.



So, while you can't buy a box of observability and sprinkle it over your monitoring tools, it's still a concept with which you ought to be familiar so you can understand which of your monitoring solutions fit the criteria.

Tips, Tricks, and Traps

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

FLAPPING (OR "SAWTOOTHING")

When an alert repeatedly triggers (a device keeps rebooting itself, a disk drive hovering on the edge of full, and processes keep deleting/creating temporary files so one moment it's over threshold, the next it's below), it's a condition known as flapping or sawtoothing.

Here are a few techniques you can use 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 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.
- Don't 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 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. The 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're 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're 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's 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 unresponsive device. 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, even 1% isn't enough to cause immediate alarm (remember, 1% remaining on a 1 terabyte drive is still 100+ gigabytes).

Instead, in some cases, you want to monitor the delta, or the rate of change. Here, you might be interested in knowing 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, the 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.
- Deduplication: One of two techniques to "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, which was closed by a human operator, or 2) a certain amount of time has elapsed.
- Alert after XX times: The second of two techniques to address flapping. In this case, the flapping is itself an indicator. If a problem occurs once, it's 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).

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 running on every device 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 monitoring 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 can run certain actions that simply don't work between two devices across a network connection.

The downside of agent-based monitoring is the great deal of effort needed for deploying, maintaining, and monitoring the health of the agent itself. In addition, a monitoring agent can cause problems on a device rather than remaining out of the way and monitoring conditions without affecting them.

Agentless, on the other hand, is generally innocuous regarding its impact on the device being monitored. Even when the technique used is a script executed on the device to be monitored, it a poorly written script would be causing the negative impact to the device, instead of the monitoring tool itself.

The other upside of agentless monitoring is the ongoing management and upkeep. Because there are no agents to patch or upgrade, all the work of maintaining the monitoring solution centers around keeping the polling engine and database healthy, which is a much smaller level of effort

Of course, the trade-off is agentless monitoring often can't accomplish as much regarding the granularity of data collected as well as the robustness of an automated response if a problem occurs. 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 job—not one of "good" versus "bad."

So What Should I Buy?

Regardless of the data you need and the techniques you're comfortable implementing, you still need a software tool to do some of the things described in this document. Every software vendor is working from the same basic playbook. That's good—you're comparing apples to apples. But it makes selecting a tool hard, because you may not know what you should look for as a differentiating factor. What, exactly, 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. In my experience, the cost to set up good monitoring is consistent regardless of which solution you choose. What changes is the way you pay for it.

At one end, you could go with a 100% DIY solution lovingly crafted from custom-written bash scripts. This might have zero-dollar cost, but a high cost in terms of internal effort to create; and an equally high cost of internal effort to maintain.

At the other end, there are turnkey solutions leveraging pre-provisioned hardware loaded into the data center as-is, with long-term consulting to ensure everything is tuned to perfection before it rolls to production. Going this route requires lots of cash, but minimal internal effort.

Obviously, those are the two extremes and I've wasted two paragraphs essentially explaining the "build vs. buy" scenario we've all understood since the first day we set foot in our first data center.

But with monitoring this choice matters because—despite a healthy selection of tools along the spectrum—there are still only a handful of technologies by which monitoring gets done. OK, maybe two handfuls.

The differences in the solution you pick comes down to the shade of lipstick vendors put on those venerable piggies.

However, "shade of lipstick" means more than just the user interface and graphing style. It's the ease of deployment, configuration, and maintenance; the flexibility of what you can DO with the tool once it's set up and configured; and, most important of all, the availability of the data (both to external systems and to other modules within the solution) once it's collected.

So... even though I rely on a particular vendor to pay my kids orthodontia, the real answer comes down to three fundamental considerations:

1. The kinds of data you'll need to have at your fingertips once monitoring is deployed.
2. The staff (quantity, availability, and skillsets) you have at your disposal to get this done
3. What you can afford to pay—as measured in dollars, and/or in hours, and/or in political capital.

Meanwhile, 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 you should NEVER trust the vendor to answer. Talk to other companies using the software.
- How much customization is needed?

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

The Mostly Un-Necessary Summary

For many IT professionals, monitoring is a checkbox item on their daily to-do list, rather than the focus of the work they do. While it's understandable for network engineers, SysAdmins, cloud practitioners, virtualization experts, InfoSec professionals, etc., to be interested in monitoring, the truth of the matter is monitoring has become a discipline within IT all to its own—as important to ensuring the success of the business as having top-notch networks, systems, storage, and security.

But without the right tools in place, monitoring can quickly devolve into a tangled mess of missing data, guesswork, long nights, empty cola cans, and regret. The first step to having the right tools in place is understanding—clearly and without marketing hyperbole—what those monitoring protocols and techniques are doing under the hood as well as what kind of data they can be expected to deliver.

Dedications and Acknowledgements

To Debbie.

When I first published this book, I said, "As with anything I have ever done that mattered, this is dedicated to my best friend and my closest confidant. Even after three decades, you still mean more to me than everything else." None of that has changed. If anything, our experiences and the challenges life has tossed our way have made me appreciate your presence in my life more than I could have imagined back when I first put those words down on paper.

For additional information, please contact SolarWinds at 866.530.8100 or email sales@solarwinds.com.
To locate an international reseller near you, visit http://www.solarwinds.com/partners/reseller_locator.aspx