# Table of Contents

**Monitoring Anti-Patterns**

<table>
<thead>
<tr>
<th>Anti-Pattern #1: Tool Obsession</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitoring Is Multiple Complex Problems Under One Name</td>
<td>6</td>
</tr>
<tr>
<td>Avoid Cargo-Culting Tools</td>
<td>8</td>
</tr>
<tr>
<td>Sometimes, You Really Do Have to Build It</td>
<td>9</td>
</tr>
<tr>
<td>The Single Pane of Glass Is a Myth</td>
<td>10</td>
</tr>
<tr>
<td>Anti-Pattern #2: Monitoring-as-a-Job</td>
<td>10</td>
</tr>
<tr>
<td>Anti-Pattern #3: Checkbox Monitoring</td>
<td>11</td>
</tr>
<tr>
<td>OS Metrics Aren't Very Useful—for Alerting</td>
<td>12</td>
</tr>
<tr>
<td>Collect Your Metrics More Often</td>
<td>13</td>
</tr>
<tr>
<td>Anti-Pattern #4: Using Monitoring as a Crutch</td>
<td>13</td>
</tr>
<tr>
<td>Anti-Pattern #5: Manual Configuration</td>
<td>14</td>
</tr>
<tr>
<td>Wrap-Up</td>
<td>15</td>
</tr>
</tbody>
</table>
Before we can start off on our journey to great monitoring, we have to identify and correct some bad habits you may have adopted or observed in your environment.

As with many habits, they start off well-meaning. After years of inadequate tools, the realities of keeping legacy applications running, and a general lack of knowledge about modern practices, these bad habits become “the way it’s always been done” and are often taken with people when they leave one job for another. On the surface, they don’t look that harmful. But rest assured—they are ultimately detrimental to a solid monitoring platform. For this reason, we’ll refer to them as anti-patterns.

An anti-pattern is something that looks like a good idea, but which backfires badly when applied.

—Jim Coplien

These anti-patterns can often be difficult to fix for various reasons: entrenched practices and culture, legacy infrastructure, or just plain old FUD (fear, uncertainty, and doubt). We’ll work through all of those, too, of course.

**Anti-Pattern #1: Tool Obsession**

There’s a great quote from Richard Bejtlich in his book *The Practice of Network Security Monitoring* (No Starch Press, 2013) that underscores the problem with an excessive focus on tools over capabilities:

Too many security organizations put tools before operations. They think “we need to buy a log management system” or “I will assign one analyst to antivirus duty, one to data leakage protection duty.” And so on. A tool-driven team will not be effective as a mission-driven team. When the mission is defined by running software, analysts become captive to the features and limitations of their tools. Analysts who think in terms of what they need in order to accomplish their mission will seek tools to meet those needs, and keep looking if their requirements aren’t met. Sometimes they even decide to build their own tools.
Many monitoring efforts start out the same way. “We need better monitoring!” someone says. Someone else blames the current monitoring toolset for the troubles they’re experiencing and suggests evaluating new ones. Fast-forward six months and the cycle repeats itself.

If you learn nothing else from this book, remember this: there are no silver bullets.

Anything worth solving takes a bit of effort, and monitoring a complex system is certainly no exception. Relatedly, there is no such thing as the single-pane-of-glass tool that will suddenly provide you with perfect visibility into your network, servers, and applications, all with little to no tuning or investment of staff. Many monitoring software vendors sell this idea, but it’s a myth.

Monitoring isn’t just a single, cut-and-dry problem—it’s actually a huge problem set. Even limiting the topic to server monitoring, we’re still talking about handling metrics and logs for server hardware (everything from the out-of-band controller to the RAID controller to the disks), the operating system, all of the various services running, and the complex interactions between all of them. If you are running a large infrastructure (like I suspect many of you are), then paying attention only to your servers won’t get you very far: you’ll need to monitor the network infrastructure and the applications too.

Hoping to find a single tool that will do all of that for you is simply delusional.

So what can you do about it?

**Monitoring Is Multiple Complex Problems Under One Name**

We’ve already established that monitoring isn’t a single problem, so it stands to reason that it can’t be solved with a single tool either. Just like a professional mechanic has an entire box of tools, some general and some specialized, so should you:

- If you’re trying to profile and monitor your applications at the code level, you might look at APM tools, or instrumenting the application yourself (e.g., StatsD).
- If you need to monitor performance of a cloud infrastructure, you might look at modern server monitoring solutions.
- If you need to monitor for spanning tree topology changes or routing updates, you might look at tools with a network focus.
In any mature environment, you’ll fill your toolbox with a set of general and specialized tools.

**The Observer Effect Isn’t a Problem**

The observer effect states that the act of observing a thing causes that thing to change. When it comes to technology, this is often taken to mean that any observation tools are going to place additional load on a system. That’s true—but it’s not actually a problem.

Let me put this one to rest now: it’s 2017, not 1999—your systems can handle the additional load, however minuscule it will be (and it will be minuscule).

Some people seem concerned about using an agent at all. Agents aren’t a bad thing: how else are you planning to get metrics out of these systems? If you’re worried about management and configuration of the agent, you should be using a configuration management tool (if you’re not, put this book down and go learn and implement config management). Agentless monitoring is extraordinarily inflexible and will never give you the amount of control and visibility you desire—bite the bullet and you’ll never look back.

I’ve found it’s common for people to be afraid of tool creep. That is, they are wary of bringing more tools into their environment for fear of increasing complexity. This is a good thing to be wary of, though I think it’s less of a problem than most people imagine.

**How Many Tools Is Too Many?**

How many monitoring tools is too many? Unfortunately, there’s no hard-and-fast rule I can give you on this. I’ve worked with companies supporting only a handful of tools for specific purposes all the way up to large enterprises with over a hundred different tools, many of them overlapping in purpose and function.

The guidance I can provide is this: as few as you need to get the job done. If you have three tools to monitor your database and they all provide the same information, you should consider consolidating. On the other hand, if all three of those tools monitor your databases and provide different information, you’re probably fine. Tool fragmentation is only a real problem when the tools don’t work together and can’t correlate their data.

My advice is to choose tools wisely and consciously, but don’t be afraid of adding new tools simply because it’s yet another tool. It’s a good thing that your network engi-
neers are using tools specialized for their purpose. It’s a good thing that your software engineers are using APM tools to dive deep into their code.

In essence, it’s desirable that your teams are using tools that solve their problems, instead of being forced into tools that are a poor fit for their needs in the name of “consolidating tools.” If everyone is forced to use the same tools, it’s unlikely that you’re going to have a great outcome, simply due to a poor fit. On the other hand, where you should be rightfully worried is when you have many tools that have an inability to work together. If your systems team can’t correlate latency on the network with poor application responsiveness, you should reevaluate your solutions.

What if you want to set some company standards on tools to prevent runaway adoption? In essence, you might have dozens of tools all doing the same thing. In such a case, you’re missing out on the benefits that come with standardization: institutional expertise, easier implementation of monitoring, and lower expenses. How would you go about determining if you’re in that situation?

It’s easy. Well, sort of: you have to start talking to people, and a lot of them. I find it helpful to start an informal conversation with managers of teams and find out what monitoring tools are being used and for what purpose. Make it clear right away that you’re not setting out to change how they work—you’re gathering information so you can help make their jobs easier later. Forcing change on people is a great way to derail any consolidation effort like this, so keep it light and informal for now. If you’re unable to get clear answers, check with accounting: purchase orders and credit card purchases for the past year will reveal both monthly SaaS subscriptions and annual licensing/SaaS subscriptions. Make sure to confirm what you find is actually in use though—you may just find tools that are no longer in use and haven’t been cancelled yet.

Avoid Cargo-Culting Tools

There is a story recounted in Richard Feynman’s book *Surely You’re Joking, Mr. Feynman!* about what Feynman dubbed cargo cult science:

In the South Seas there is a cargo cult of people. During the war they saw airplanes land with lots of good materials, and they want the same thing to happen now. So they’ve arranged to imitate things like runways, to put fires along the sides of the runways, to make a wooden hut for a man to sit in, with two wooden pieces on his head like headphones and bars of bamboo sticking out like antennas—he’s the controller—and they wait for the airplanes to land. They’re doing everything right. The form is perfect. It looks exactly the way it looked before. But it doesn’t work. No airplanes land.

So I call these things cargo cult science, because they follow all the apparent precepts and forms of scientific investigation, but they’re missing something essential, because the planes don’t land.

Over the years, this observation from the science community has become applied to software engineering and system administration: adopting tools and procedures of
more successful teams and companies in the misguided notion that the tools and procedures are what made those teams successful, so they will also make your own team successful in the same ways. Sadly, the cause and effect are backward: the success that team experienced led them to create the tools and procedures, not the other way around.

It's become commonplace for companies to publicly release tools and procedures they use for monitoring their infrastructure and applications. Many of these tools are quite slick and have influenced the development of other monitoring solutions widely used today (for example, Prometheus drew much inspiration from Google's internal monitoring system, Borgmon).

Here's the rub: what you don't see are the many years of effort that went into understanding why a tool or procedure works. Blindly adopting these won't necessarily lead to the same success the authors of those tools and procedures experienced. Tools are a manifestation of ways of working, of assumptions, of cultural and social norms. Those norms are unlikely to map directly to the norms of your own team.

I don't mean to discourage you from adopting tools published by other teams—by all means, do so! Some of them are truly amazing and will change the way you and your colleagues work for the better. Rather, don't adopt them simply because a well-known company uses them. It is important to evaluate and prototype solutions rather than choosing them because someone else uses them or because a team member used them in the past. Make sure the assumptions the tools make are assumptions you and your team are comfortable with and will work well within. Life is too short to suffer with crummy tools (or even great tools that don't fit your workflow), so be sure to really put them through their paces before integrating them into your environment. Choose your tools with care.

**Sometimes, You Really Do Have to Build It**

When I was growing up, I loved to go through my grandfather’s toolbox. It had every tool imaginable, plus some that baffled me as to their use. One day, while helping my grandfather fix something, he suddenly stopped, looking a little perplexed, and began rummaging through the toolbox. Unsatisfied, he grabbed a wrench, a hammer, and a vice. A few minutes later he had created a new tool, built for his specific need. What was once a general-purpose wrench became a specialized tool for solving a problem he never had before. Sure, he could have spent many more hours solving the problem with the tools he had, but creating a new tool allowed him to solve a particular problem in a highly effective manner, in a fraction of the time he might have spent otherwise.

Creating your own specialized tool does have its advantages. For example, one of the first tools many teams build is something to allow the creation of AWS EC2 instances quickly and with all the standards of their company automatically applied. Another
example, this one monitoring-related, is a tool I once created: while working with SNMP (which we’ll be going into in Chapter 9), I needed a way to comb through a large amount of data and pull out specific pieces of information. No other tool on the market did what I needed, so with a bit of Python, I created a new tool suited for my purpose.

Note that I’m not suggesting you build a completely new monitoring platform. Most companies are not at the point where the ground-up creation of a new platform is a wise idea. Rather, I’m speaking to small, specialized tools.

The Single Pane of Glass Is a Myth

Every Network Operations Center (NOC) I’ve been in has had gargantuan monitors covering the wall, filled with graphs, tables, and other information. I once worked in a NOC (pronounced like “knock”) myself that had six 42” monitors spanning the wall, with constant updates on the state of the servers, network infrastructure, and security stance. It’s great eye candy for visitors.

However, I’ve noticed there can often be a misconception around what the single pane of glass approach to monitoring means. This approach to monitoring manifests as the desire to have one single place to go to look at the state of things. Note that I didn’t say one tool or one dashboard—this is crucial to understanding the misconception.

There does not need to be a one-to-one mapping of tools to dashboards. You might use one tool to output multiple dashboards or you might even have multiple tools feeding into one dashboard. More likely, you’re going to have multiple tools feeding multiple dashboards. Given that monitoring is a complex series of problems, attempting to shoehorn everything into one tool or dashboard system is just going to hamper your ability to work effectively.

Anti-Pattern #2: Monitoring-as-a-Job

As companies grow, it’s common for them to adopt specialized roles for team members. I once worked for a large enterprise organization that had specialized roles for everyone: there was the person who specialized in log collection, there was the person who specialized in managing Solaris servers, and another person whose job it was to create and maintain monitoring for all of it. Three guesses which one was me.

At first glance, it makes sense: create specialized roles so people can focus on doing that function perfectly, instead of being a generalist and doing a mediocre job on everything. However, when it comes to monitoring, there’s a problem: how can you build monitoring for a thing you don’t understand?

Thus, the anti-pattern: monitoring is not a job—it’s a skill, and it’s a skill everyone on your team should have to some degree. You wouldn’t expect only one member of
your team to be the sole person familiar with your config management tool, or how to manage your database servers, so why would you expect that when it comes to monitoring? Monitoring can't be built in a vacuum, as it's a crucial component to the performance of your services.

As you move along your monitoring journey, insist that everyone be responsible for monitoring. One of the core tenets of the DevOps movement is that we're all responsible for production, not just the operations team. Network engineers know best what should be monitored in the network and where the hot spots are. Your software engineers know the applications better than anyone else, putting them in the perfect position to design great monitoring for the applications.

Strive to make monitoring a first-class citizen when it comes to building and managing services. Remember, it's not ready for production until it's monitored. The end result will be far more robust monitoring with great signal-to-noise ratio, and likely far better signal than you've ever had before.

There is a distinction that must be made here, of course: the job of building self-service monitoring tools as a service you provide to another team (commonly called an observability team) is a valid and common approach. In these situations, there is a team whose job is to create and cultivate the monitoring tools that the rest of the company relies on. However, this team is not responsible for instrumenting the applications, creating alerts, etc. The anti-pattern I want to caution you against isn't having a person or team responsible for building and providing self-service monitoring tools, but rather, it's having your company shirk the responsibility of monitoring at all by resting it solely on the shoulders of a single person.

**Anti-Pattern #3: Checkbox Monitoring**

When people tell me that their monitoring sucks, I find that this anti-pattern is usually at the center of it all.

Checkout monitoring is when you have monitoring systems for the sole sake of saying you have them. Perhaps someone higher up in the organization made it a requirement, or perhaps you suddenly had specific compliance regulations to meet, necessitating a quick monitoring deployment. Regardless of how you got here, the result is the same: your monitoring is ineffective, noisy, untrustworthy, and probably worse than having no monitoring at all.

How do you know if you've fallen victim to this anti-pattern? Here are some common signs:

- You are recording metrics like system load, CPU usage, and memory utilization, but the service still goes down without your knowing why.
• You find yourself consistently ignoring alerts, as they are false alarms more often than not.
• You are checking systems for metrics every five minutes or even less often.
• You aren’t storing historical metric data (I’m looking at you, Nagios).

This anti-pattern is commonly found with the previous anti-pattern (monitoring-as-a-job). Since the person(s) setting up monitoring doesn’t completely understand how the system works, they often set up the simplest and easiest things and check it off the to-do list.

There are a few things you can do to fix this anti-pattern.

**What Does “Working” Actually Mean? Monitor That.**

To fix this problem, you first need to understand what it is you’re monitoring. What does “working” mean in this context? Talking to the service/app owner is a great place to start.

Are there high-level checks you can perform to verify it’s working? For example, if we’re talking about a webapp, the first check I would set up is an HTTP GET / . I would record the HTTP response code, expect an HTTP 200 OK response, specific text to be on the page, and the request latency. This one check has given me a wealth of information about whether the webapp is actually working. When things go south, latency might increase while I continue to receive an HTTP 200 response, which tells me there might be a problem. In another scenario, I might get back the HTTP 200, but the text that should be on the page isn’t found, which tells me there might be a problem.

Every service and product your company has will have these sorts of high-level checks. They don’t necessarily tell you what’s wrong, but they’re great leading indicators that something could be wrong. Over time, as you understand your service/app more, you can add more specific checks and alerts.

**OS Metrics Aren’t Very Useful—for Alerting**

Early in my career as a systems administrator, I went to my lead engineer and told him that the CPU usage on a particular server was quite high, and asked what we should do about it. His response was illuminating for me: “Is the server still doing what it’s supposed to?” It was, I told him. “Then there’s not really a problem, is there?”

Some services we run are resource-intensive by nature and that’s OK. If MySQL is using all of the CPU consistently, but response times are acceptable, then you don’t really have a problem. That’s why it’s far more beneficial to alert on what “working” means as opposed to low-level metrics such as CPU and memory usage.
That isn’t to say these metrics aren’t useful, of course. OS metrics are critical for diagnostics and performance analysis, as they allow you to spot blips and trends in underlying system behavior that might be impacting performance. 99% of the time, they aren’t worth waking someone up over. Unless you have a specific reason to alert on OS metrics, stop doing it.

Collect Your Metrics More Often

In a complex system (like the one you are running), a lot can happen in a few minutes, or even a few seconds. Let’s consider an example: imagine latency between two services spikes every 30 seconds, for whatever reason. At a five-minute metric resolution, you would miss the event. Only collecting your metrics every five minutes means you’re effectively blind. Opt for collecting metrics at least every 60 seconds. If you have a high-traffic system, opt for more often, such as every 30 seconds or even every 10 seconds.

Some people have argued that collecting metrics more often places too much load on the system, which I call baloney. Modern servers and network gear have very high performance and can easily handle the minuscule load more monitoring will place on them.

Of course, keeping high-granularity metrics around on disk for a long period of time can get expensive. You probably don’t need to store a year of CPU metric data at 10-second granularity. Make sure you configure a roll-up period that makes sense for your metrics.¹

The one caveat with this is that many older network devices often have very low performance available to the management cards, causing them to fall over when hit with too many requests for monitoring data (I’m looking at you, Cisco). Be sure to test them in a lab before increasing the polling interval for these.

Anti-Pattern #4: Using Monitoring as a Crutch

I once worked with a team that ran a legacy PHP app. This app had a large amount of poorly written and poorly understood code. As things tended to break, the team’s usual response was to add more monitoring around whatever it was that broke. Unfortunately, while this response seems at first glance to be the correct response, it does little to solve the real problem: a poorly built app.

Avoid the tendency to lean on monitoring as a crutch. Monitoring is great for alerting you to problems, but don’t forget the next step: fixing the problems. If you find yourself with a finicky service and you’re constantly adding more monitoring to it, stop

¹ Consult the documentation for your metrics tool on roll-up configuration and best practices.
and invest your effort into making the service more stable and resilient instead. More monitoring doesn’t fix a broken system, and it’s not an improvement in your situation.

**Anti-Pattern #5: Manual Configuration**

I’m sure we all can agree that automation is awesome. That’s why it’s surprising to me how often monitoring configuration is manual. The question I never want to hear is “Can you add this to monitoring?”

---

**Monitoring Cloud Architectures Versus Traditional Ones**

Monitoring cloud-based architectures differs from traditional (aka static) architectures in one big way: you are monitoring entire *classes* of things, rather than individual things. Monitoring becomes an exercise in analyzing the aggregate of entire groups of systems rather than one or two. Automation is crucial to successfully monitoring a cloud-native architecture.

Your monitoring should be 100% automated. Services should self-register instead of someone having to add them. Whether you’re using a tool such as Sensu that allows for instant self-registration and deregistration of nodes, or using Nagios coupled with config management, monitoring ought to be automatic.

The difficulty in building a well-monitored infrastructure and app without automation cannot be overstated. I’m often called on to consult on monitoring implementations, and in most cases, the team spends more time on configuration than on monitoring. If you cannot quickly configure new checks or nodes, building better monitoring becomes frustrating. After a while, you’ll just stop bothering. On the other hand, if it takes only a few minutes to add new checks for every web server in your fleet, you won’t be so hesitant to do more of it.

---

**Runbook Abuse**

A special note on runbooks, which I talk about in more detail in Chapter 3: a runbook can often be a symptom of inadequate automation. If your runbook is simply a list of steps to take (“Run this command, check this information, run this other command”), then you need more automation. If the alert the runbook is referencing can be solved by simply going through a list of steps, consider automating those steps and having your monitoring tool execute that before alerting you.
Wrap-Up

We learned about five common anti-patterns in monitoring in this chapter:

- Tool obsession doesn’t give you better monitoring.
- Monitoring is everyone’s job, not a single role on the team or a department.
- Great monitoring is more than checking the box marked “Yep, we have monitoring.”
- Monitoring doesn’t fix broken things.
- Lack of automation is a great way to ensure you’ve missed something important.

Now that you know the monitoring anti-patterns to watch out for and how to fix them, you can build positive monitoring habits. If you were to do nothing but fix these five problems in your environment, you’d be in good shape. Of course, who wants to settle for good when they can be great? And for that, we’ll need to talk about the inverse of the anti-pattern: the design pattern.
About the Author

Mike Julian is a consultant who helps companies build better monitoring for their applications and infrastructure. He is the Editor of Monitoring Weekly, an online publication about all-things-monitoring. Mike has previously worked as an operations/DevOps engineer for Taos Consulting, Peak Hosting, Oak Ridge National Lab, and others.

Mike is from Knoxville, Tennessee, and lives in San Francisco, California. Outside of work, he spends his time driving mountain roads in a classic BMW, reading, and traveling.

You find find Mike at:

- https://www.mikejulian.com
- Aster Labs
- Monitoring Weekly

Colophon

The animal on the cover of Practical Monitoring, the Bengal monitor (Varanus bengalensis), is a diurnal monitor lizard found across southern and western Asia. Dwelling primarily in low-elevation areas, they are adaptable enough to survive in a variety of habitats with varying degrees of moisture and vegetation, from forests, to bogs, to agricultural zones. They are known to grow to a length of 175 cm (100 cm of which comprise the tail), and have few predators in adulthood, aside from humans, who hunt monitors for their meat and fat, as well as their skin, which is used as the drum head for the kanjira, a hand-percussion instrument.

Though subsisting primarily on arthropods, Bengal monitors are known to eat most creatures small enough to overpower, including a wide variety of vertebrates, birds, and fish, as well as occasional fruits and vegetables. The Bengal mating season lasts from June through September, and female monitors bury their clutch of eggs, taking care to compact the soil over the nest while digging poorly-concealed decoy nests in the surrounding area.

Monitor lizards have a higher metabolism than most reptiles and spend most of their waking moments engaged in eating and physical activity. They are strong swimmers, capable of high speeds on the ground, in the water, and while scaling trees, where they take the eggs from birds’ nests and consume sleeping bats.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to animals.oreilly.com.
The cover image is from John George Wood’s *Animate Creation*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag’s Ubuntu Mono.