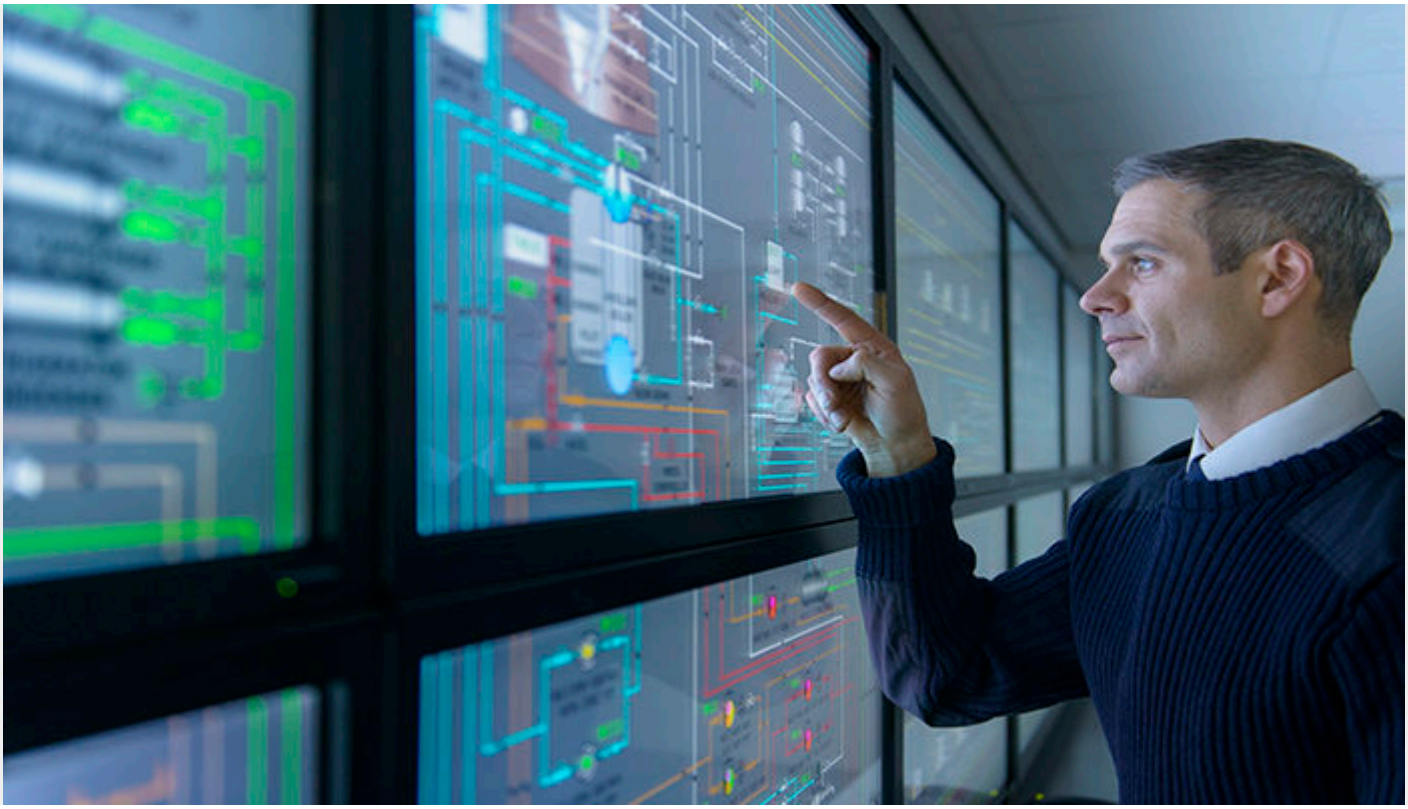
TRACING VS LOGGING VS MONITORING: WHAT'S THE DIFFERENCE?



Whether you're a systems administrator or a developer, you'll soon want to understand how your software works. From a single microservice to a vast, monolithic system, logging, tracing, and monitoring are all ways to help ensure correctness in your system, to track what may have gone wrong when problems arise, and to improve the overall functionality.

Importantly, logging, tracing, and monitoring aren't different words for the same process. They're each functioning in a unique way. Even if some tools or technologies overlap, each process provides a different outcome to your IT environment. Let's take a look.

What is logging?

The purpose of logging is to track error reporting and related data in a centralized way. Logging should be used in big applications and it can be put to use in smaller apps, especially if they provide a crucial function. The term logging can refer both to the practice of event logging or to the actual log files that result.

Log files can show any discrete event within an application or system, such as a failure, and error, or a state transformation. When something inevitably goes wrong, such transformations in state help indicate which change actually caused an error.

Logging is primarily deployed and used by system administrators on the operational level,

intentionally providing a high-level view. The most successful log files are not noisy; they shouldn't contain extraneous or distracting information. Instead log files should log only what is absolutely necessary, such as actionable items.

Of these action-related items, you may have two types of data: data for us humans that alerts or warns of a panic situation (enough to begin the investigation but not an overwhelming amount), as well as structured data for machines. (Some debate whether this machine-level data is necessary, but security is a good case use.)

Consider that logging should tell a compelling story, but as succinctly as possible. When choosing what to log, consider who is using the logs (typically sysadmins), and whether logging helps only with preventative measures or with ongoing pursuits. Are all system errors equal, or does a warning in a particular area serve as a warning for a critical failure elsewhere?

Other characteristics of successful logs: output is often standards-based, logs are localized, and new events need not be agile.

Logging too much data can be distracting and a poor use of resources. Indeed, transferring, storing and parsing logs is expensive, so minimizing what the log files contains can minimize cost and resources. Still, logging is king, especially when it comes to traditional monolithic architectures.

What is tracing?

Where logging provides an overview to a discrete, event-triggered log, tracing encompasses a much wider, continuous view of an application. The goal of tracing is to following a program's flow and data progression. As such, there is a lot more information at play; tracing can be a lot noisier of an activity than logging – and that's intentional.

In many instances, tracing represents a single user's journey through an entire app stack. Its purpose isn't reactive, but instead focused on optimization. By tracing through a stack, developers can identify bottlenecks and focus on improving performance.

When a problem does occur, tracing allows you to see how you got there: which function, the function's duration, parameters passed, and how deep into the function the user could get. A common tracing tool is the Profiling API in .NET.

In an ideal world, every function has tracing enabled. But, the amount of resulting data can be too much to sort, though cloud technology is certainly helping tracing become a realistic option for more time. Unlike logging, localization is not a concern, but new messages do need to be agile.

Other drawbacks include complex layers, abundance of implementation code, and a push model, a common design, which can affect applications. To illustrate this, tracing libraries that intend to simplify tracing as a practice often wind up being more complicated than the code they are serving. Sometimes, tracing is best for microservices.

Because of the data involved, tracing can be an expensive endeavor. Before stepping into tracing, remember that it is not a requirement. You'll want to consider whether the added complexity is warranted, what value will it bring? You may fall into a trap of optimizing prematurely, or you may be able to scale horizontally and avoid such optimization for a time.

What is monitoring?

While monitoring may be a casual term that can be applied to tracing or logging or a number of other activities, in this context, monitoring is much more specific: instrumenting an application and then collecting, aggregating, and analyzing metrics to improve your understanding of how the system behaves.

This type of monitoring is primarily diagnostic – for instance, alerting developers when a system isn't work as it should. In an ideal world where cost isn't a problem, you could instrument and monitor all of your services

Monitoring systems are the best way to begin employing metrics. Such systems handle storage, aggregation, visualization, and even automated responses. These monitoring systems are surprisingly affordable, though they do rely heavily on data. With companies embracing cloud and data, the more data you have, the more beneficial monitoring can be.

Choosing tracing, logging, or monitoring

Certainly, companies don't have to deploy only one tool, as each process has its own goals and outcomes. Often logging is the first step, held up by many as a requirement. Tracing or monitoring, at least for now, may be beneficial but not necessities; as you grow and need more functionality, one or both can be useful.

It is important to remember, however, that each of the three are not, in and of themselves, solutions. Instead, logging, tracing, and monitoring are simply proxy representations of the actual action software performs. They are not what the software is or does – they are simply tools to understand how a system behaves.