**The Three Essentials for Achieving Full-Stack Observability**

Let’s imagine a common scenario that plays out regularly at many companies today.

An engineer, Sally, gets a page in the middle of the night about an outage affecting what digital customers experience on the website. So every second counts, but from the information in the pager alert on her mobile phone, she cannot tell what the issue is. She rolls out of bed, and logs into her performance monitoring tool to find out what’s wrong.

To her surprise, everything she can see without scrolling is showing up with an issue. Every one of the more than 2,000 services for which she’s responsible is appearing red with performance degradation. This is bad—really bad. Sally starts to scroll, but she can’t get an accurate count of how many are affected, nor can she quickly tell if any of these services have something in common, like a cluster, a framework, team, or related services. She also can’t quickly see what metrics are registering an unusual reading.

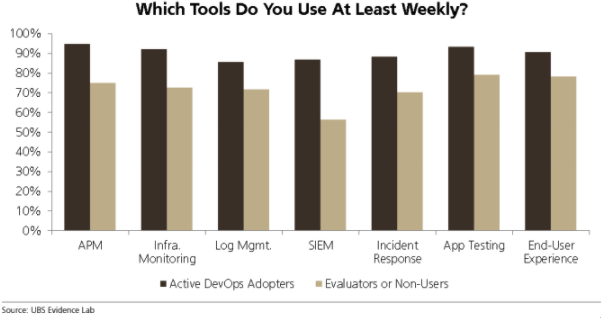
So, one by one, she navigates from her application monitoring tool to a log management system to an infrastructure monitoring tool to look for clues and commonalities. She even checks the real-user monitoring tool. This is time-consuming, ripe for human error, and results in lost sleep for Sally, and dissatisfied customers and money out the door for her employer. But this is the reality of how many companies experience what they call “observability.” They’re confusing monitoring with true observability.

**Siloed vs. full-stack observability**

What many companies are missing is that the experience of observability—full-stack, end-to-end, seeing-the-whole-IT-stack visibility—can’t and shouldn’t be siloed by individual monitoring tools. Developer roles may be consolidating—about 55% of the developers globally who responded to [Stack Overflow’s research](https://insights.stackoverflow.com/survey/2020#overview) identify as “full-stack” developers in 2020 vs. 29% in 2015. But a full-stack developer, especially one whose organization has adopted a DevOps culture, is likely using multiple tools with multiple datasets to gain what we at New Relic define as observability: the ability to understand the behavior of your complex digital system.

And by “complex digital system,” we mean all the code, all the services, the infrastructure, the user behavior, the logs, metrics, events, and traces you collect from across your entire landscape. Sally’s microservices and distributed systems may give her more agility, scalability, and efficiency for customer-facing applications and critical workloads, but they make it increasingly difficult for her to easily view the big picture and gain true observability.

And organizations aren’t at fault, really—as their entire IT estates grew and became more complex, so did the number of monitoring tools. But none of those tools deliver a single source of truth into the end-to-end performance of the full stack. A [UBS Evidence Lab](https://www.ubs.com/global/en/investment-bank/evidence-lab/overview.html) report validates this. Respondents in organizations that have adopted a DevOps culture are using an average of four to five tools to perform their job daily—everything from APM to log management to SIEM.



But juggling multiple monitoring tools to get the full picture across your software systems or to find and fix problems creates blind spots, increases toil, and makes it harder to diagnose issues that may be impacting different parts of your estate or multiple layers of your application stack. In short, they gain siloed monitoring, or maybe they call it “observability,” but in our book it isn’t observability if it isn’t end-to-end observability.

**What is end-to-end observability?**

So let us clarify what we mean when we refer to end-to-end, or full-stack, observability:

Full-stack observability is every engineer’s **single source of truth** as they troubleshoot, debug, and optimize performance **across their entire stack**. Users can find and fix problems **faster** in one **unified experience** that provides **connected** context and surfaces meaningful analytics—from logs, infrastructure and applications, distributed tracing, serverless functions, all the way into end-user experience—without having to onboard new tools or switch between them.

For our hypothetical Sally engineer, executing on and experiencing this type of observability depends on prioritizing three system attributes:

**1. Connected context.** When Sally accesses metrics about the health of one of her 2,000 services, she should be able to see how that service affects other services or parts of the distributed system, how those workloads are affected by the Kubernetes cluster that hosts them, and vice versa. And she should be able to understand how the issue between the cluster and the app is affecting the end-user experience on her company’s website, e-commerce portal, or mobile app, all from one system.

Connected context is a reality that learning tools provider [Chegg](https://newrelic.com/resources/case-studies/Chegg" \t "_blank) enjoys. It gained a consolidated view of log messages in context with event and trace data to assemble a complete picture of an incident. Regardless of each engineer’s focus, whether back-end, system administrator, or web developer, they need to receive immediate context across a full stack, which depends on the next attribute.

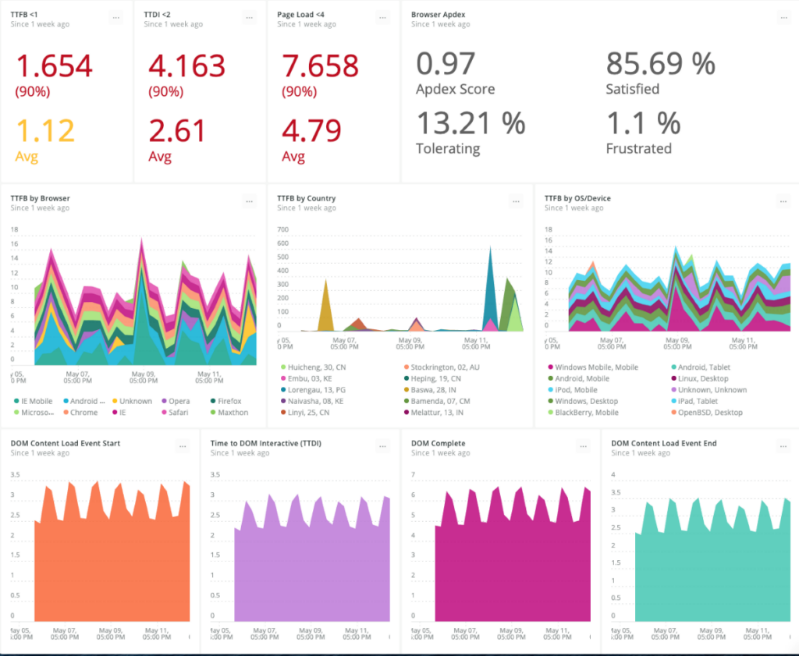
**2.** **A single (open) source of truth**. That means one place to store, alert on, and analyze operational data. Sally would need a platform that can ingest metrics, events, logs, and traces from any source, whether from proprietary or open source agents, or via APIs and built-in instrumentation. And that one place would need to be powerful enough to scale for handling the ingest load on her company’s biggest days. Often, companies prioritize only one type of telemetry, such as logs or metrics, or they may sample data from only a subset of systems, applications, or instances. Both lead to holes in observability and slow troubleshooting.

The ops manager at publishing and analytics company [Elsevier](https://newrelic.com/resources/case-studies/elsevier)describes this situation aptly: “I would get a 3 a.m. call about a problem, and the development engineer would tell me the application was performing perfectly, the network engineer would tell me the network was fine, and the infrastructure engineer would tell me that utilization was fine. But things were not fine, and the real challenge stemmed from the fact that they were looking at three different control planes.”

Full-stack observability relies on ingesting any telemetry data you want without worrying how to scale, or building a costly system for peak scale, or swivel-chairing between multiple tools.

**3. Easier, faster exploration**. So let’s assume Sally’s company gives her a way to ingest all metrics, events, logs, and traces—from anywhere and everywhere across the company’s IT stack. And that the full-stack observability system adds context to that data, so Sally understands interdependencies and up/downstream effects of issues. She’s looking at one screen to see all of this.

Think about that—all performance data on one screen, from everywhere, in real time. That screen needs to have some pretty innovative design. Because for Sally and her team to efficiently traverse large, complex, distributed systems and quickly understand and prioritize any issue, she’ll need intuitive visualizations that require zero configuration. The entire purpose of full-stack, end-to-end observability is to enable engineers to explore and identify system issues in an instant and troubleshoot and fix them before they become a problem for customers. Speed is essential to deliver benefits of lower mean time to resolution and higher uptime. Developers should be able to innovate and chaos test with confidence, knowing that the change they’re making won’t break the system. These are the benefits of full-stack observability.



Chegg’s digital experience dashboard gives the company a single dynamic view of aggregated data over the entire Chegg portfolio, as well as the ability to filter information to focus on a single product.

Sally’s dashboard needs to let her easily explore large systems with point-and-click filtering and grouping for all components that make up her distributed system—applications, infrastructure, serverless functions, third-party integrations, and so on. It shows her where anomalies are occurring, and what changes may be contributing to those anomalies. She immediately sees how issues across the system are related, and any commonalities that exist. Saved views give her team added efficiency and collaboration when troubleshooting. Ultimately, what Sally should see when she awakes from a page at 3 a.m. is an interface so intuitive and modern that it can serve as every SRE’s and IT team’s daily real-time dashboard for understanding what’s happening across their entire environment.

The challenge of full-stack observability for many companies is the aggregation of every type of telemetry data. Engineers love their tools—especially those in a DevOps culture. Any platform delivering end-to-end observability needs to win over engineers and illustrate immediate and greater benefit than their favorite tool. Perhaps promising them more sleep is one way to motivate the change.