



The Complete Developer Takeover

Time to Integrate DevOps, Application Performance Management, & the Developer In the Team of the Future


| v1.1 | JUN-JUL 2016

Build/better

The Complete Developer Takeover

Time to Integrate DevOps, Application Performance Management, & the Developer In the Team of the Future

- s.1 The World of Application Monitoring is Evolving [p.4](#)
- s.2 Support Production Apps the DevOps Way [p.8](#)
- s.3 4 Ways the Cloud Has Influenced App Troubleshooting [p.13](#)
- s.4 Application Monitoring vs. Application Performance Management (APM) [p.21](#)
- s.5 3 Data Sets That Bring Context to Exceptions [p.24](#)



The DevOps movement is not the latest and greatest development fad. DevOps is an essential part of supporting production applications in the most efficient way possible. It's a movement that proves developers need to be closer to their application code regardless of what environment it may journey off to.

Without an agreed-upon process between your development and IT teams, supporting your production applications can become slow and frustrating for both sides. The goal is always rapid application improvement for developers. This guide will help you navigate the integration of DevOps and application performance management, allowing developers to build better applications faster.

The World of Application Monitoring is Evolving

s.1

“Perfection would be a fatal flaw for evolution”

In technology, as in life, everything evolves, grows, develops, mutates and eventually goes away, being replaced with improved versions. In the past several years, application development has evolved and applications have come to play a more central part in businesses of all kinds (Walmart’s sophisticated inventory management software is a good example of this).

Several drivers contributing to this evolution have increased application complexity. It is key for development teams to understand and evolve with these drivers.



What's Driving the Evolution of Application Monitoring?

Cloud: While simplifying the development, deployment and economy of scale of applications, the use of the cloud also added a range of elements that did not exist before (varying numbers of servers, elasticity, PaaS, etc.). At the same time, development's visibility into application performance and their ability to replicate and troubleshoot issues has been reduced.

Agile Development: The ever-increasing demand for new features has created a need for rapid, agile development. Agile increases the velocity and number of application releases. The natural result is application instability and more bugs. This challenges developers' abilities to build great applications. Instead, it results in more quick fixes, reduced documentation and reduced control.

Outsourcing: Utilizing teams of developers worldwide has created cost-effective ways to speed up development. It has also decentralized an organization's ability to easily troubleshoot problems. Today's complex applications have many different components working in parallel, which requires a wider variety of expertise to keep working as designed. As companies seek to optimize labor costs, developers are expected to be more involved and the management tools they use are expected to become more intelligent.

The evolution of applications has also driven change in application monitoring tools. These provide the data and intelligence to assist with identifying, defining, detailing, and troubleshooting application issues.

1st Generation – Watching Infrastructure

In the early stages of server/client application monitoring, tools focused on infrastructure: server uptime, server load, network equipment, and storage. Important logs and errors were simply dumped to text files. When an issue occurred, the operation or support engineer had to view the data in tools like Excel and analyze it manually. This was a tedious approach to discovering the root cause of an issue.

2nd Generation – Gathering Information

The second generation of application monitoring tools were point solutions—each answering a different need that was not covered by traditional monitoring tools. Among these were log management tools, error aggregation tools, notification tools, APM (Application Performance Management) tools, website monitoring tools, transaction monitoring tools, and many others. An engineer or support manager could get a lot more information than with the previous approach—and a lot faster. However, trying to correlate problems and train an entire IT team on a wide variety of tools was a daunting task.

Acquiring all of these tools cost thousands of dollars a month, even for small IT projects. Deploying, supporting, and maintaining these tools also required significant time from busy system administrators.

Besides cost, the result was “death by tools,” where, it became unwieldy to use all available capabilities. At some point, engineers retreated to 1st generation tools.

3rd Generation – Contextual Intelligence

Second generation issues triggered the need for 3rd generation solutions for application monitoring. Now the goal is to unify various point products into an integrated platform that can correlate information, providing context and intelligence around what happened, when, where, how often, and (the ever elusive) why.

While earlier generations of monitoring fell to the operations engineer, the application developer must have access to the infrastructure, monitoring tools and the resulting monitoring data. Supporting today's complex and ever changing web applications requires that developers be much more involved in application support in general. This drives the need for 3rd generation solutions to serve both development teams and operational functions.

Getting to Contextual Intelligence

As logs, errors, application performance, server performance, database performance, and custom application metrics become more integrated and “aware” of each other, users of 3rd generation platforms have access to a more complete picture.

Application errors do not need to be a few lines with limited information. They should be seen as a collection of data including: log data, stack traces, web requests, process data, headers, relevant variables affected, and server performance metrics at the time an error occurred. This gives a more complete picture to developers and makes diagnostics faster and more efficient.

In the past, server load may or may not have been easily correlated to an issue like web page performance—for example. With a platform that is application aware, it is easier to see that these things are connected, significantly reducing time-to-resolution.

Until now, the rapid evolution of application development has outpaced an organization’s ability to support and troubleshoot those applications. With 3rd generation tools and approaches to how applications are built and maintained, development teams can have greater power over their applications throughout the entire lifecycle.



Support Production Apps the DevOps Way

Software applications are more complex than ever. If your application is slow or down for even a few minutes, thousands of customers can be exposed in an instant. Building, testing, and maintaining applications must be managed in lockstep. Preventing and fixing application problems quickly, or from ever occurring in the first place, is critical if you want to keep your outage from trending on Twitter.



The DevOps Movement Begins

For developers in larger organizations, ensuring early success of a new application starts with keeping IT operations in the loop. It has been common practice for software development teams to work for months on a new project, throwing it at the IT operations team just in time to deploy. The [health-care.gov debacle](#) is a perfect example. The DevOps movement began as a way to address scenarios like this. In theory, DevOps fosters a culture where development and IT operations work together across the lifecycle of an application so deployments go smoothly.

Long-term success of your application depends on proper maintenance and monitoring of your applications as you continue to roll out new features. The rise of agile development has increased deployments, becoming a weekly or monthly cycle at most companies. A constant stream of new features is awesome, but with that comes an increase in bugs, performance problems, and instability. This rate of change, coupled with the overall complexity of today's applications, makes them increasingly difficult to support.



Application Support: “It Takes a Village”

Collaboration between development and operational functions is critical to keeping applications running and customers happy. If you have a small development team handling all the production support issues, they will have little time to architect the innovative tools and features that make an organization competitive.

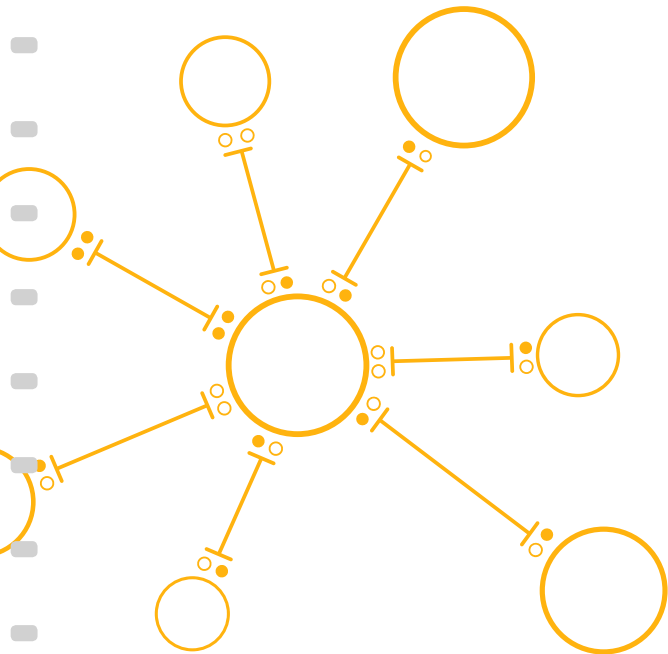
The keys to supporting production applications are (1) create cleaner code from the start, and (2) remove bottlenecks through visibility and involvement between teams, across the application lifecycle. Operations needs to make sure that the development team has visibility of basic troubleshooting information like errors, logs, and key metrics, while ensuring security. Otherwise, developers work blindly to fix basic bugs and improve application performance. Without appropriate tools, developers must rely on others for help, which will reduce efficiency and innovation. It may “take a village” to build great applications, but it shouldn’t take an eternity.

Working across functions makes sense, but IT operations may express concerns over granting access to production data without going through proper release processes. Fortunately, developers do not need administrator level access to servers, they simply need visibility to aggregated performance data. There are safe and simple ways to communicate across environments, allowing developers enough access to quickly find the root cause of an issue, while avoiding security risks or creating new issues in the process.

DevOps Should Include Production Support

The goal of DevOps is to create collaboration and improve the working relationships between development and operations. The DevOps movement initially focused on software deployment and continuous delivery. However, support of production applications is another, equally important component in the application lifecycle that is often overlooked. To solve application problems quickly, developers and operations need to stop pointing fingers and work as a team. To effectively build and support applications, developers need access to a variety of data:

- Visibility and notifications around all application errors
- Access to centralized aggregated log files for viewing and searching
- Basic server utilization trends and stats. (e.g. CPU, memory, etc.)
- Recording and alerting of key application metrics (KPIs)
- Tracking of application web page load times
- Ability to access the application database and run test queries

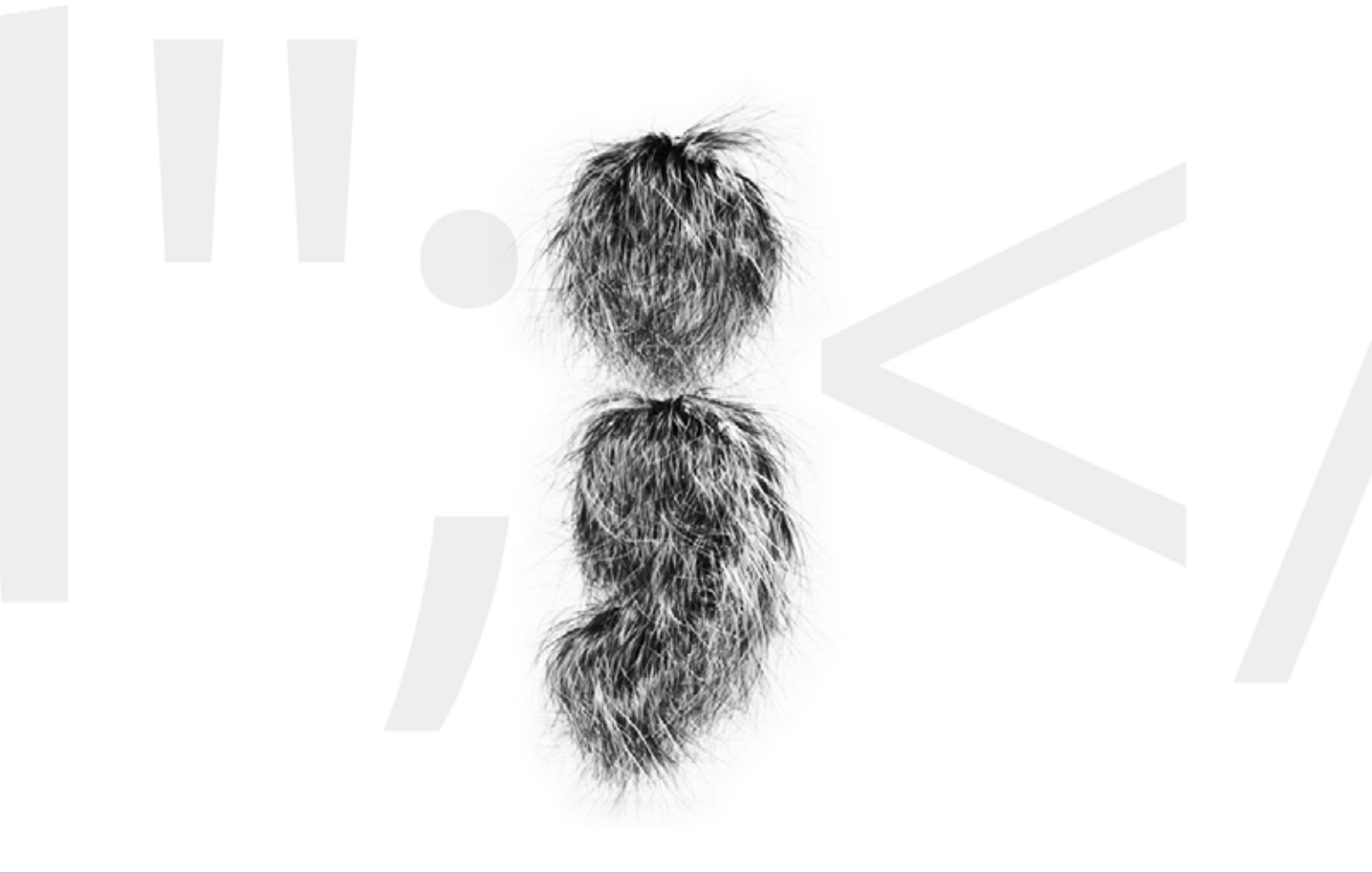


Using multiple, unrelated tools to track this data makes it difficult to correlate and identify issues. It is also difficult to train numerous team members effectively on a wide variety of tools (not to mention the expensive).

There are solutions that combine these features into one dashboard, so solving issues can be done quickly and with minimal training. Stackify, for example, gives developers access to all the data they need to monitor and troubleshoot applications—through development, testing, and production—without unnecessary access to every server.

With tools like this, you can see where the application is deployed, the current health, performance stats, recent errors, full logs, and key metrics from one application dashboard. This makes it easy for companies to embrace application support from a DevOps perspective.





Bugs get hairy fast

Get Prefix—the free, killer profiler—to find them first

[Download Now—Free!](#)

For .NET & Java web apps

4 Ways the Cloud Has Influenced App Troubleshooting

The rise of cloud computing has ushered in an era of unprecedented productivity for developers over the past several years. For those who have embraced this new world order, gone are the days of long lead times for hardware procurement and installation, architecture defined by slow-moving hardware upgrades, hardware-constrained scalability and flexibility, and a world where only sys admins have access to the infrastructure. But, as the barriers between development and delivery disappear, new challenges have emerged that can disrupt the lives of developers and slow down delivery of new products and features, giving back some of the efficiencies the Software-Defined Data Center (SDDC) created.

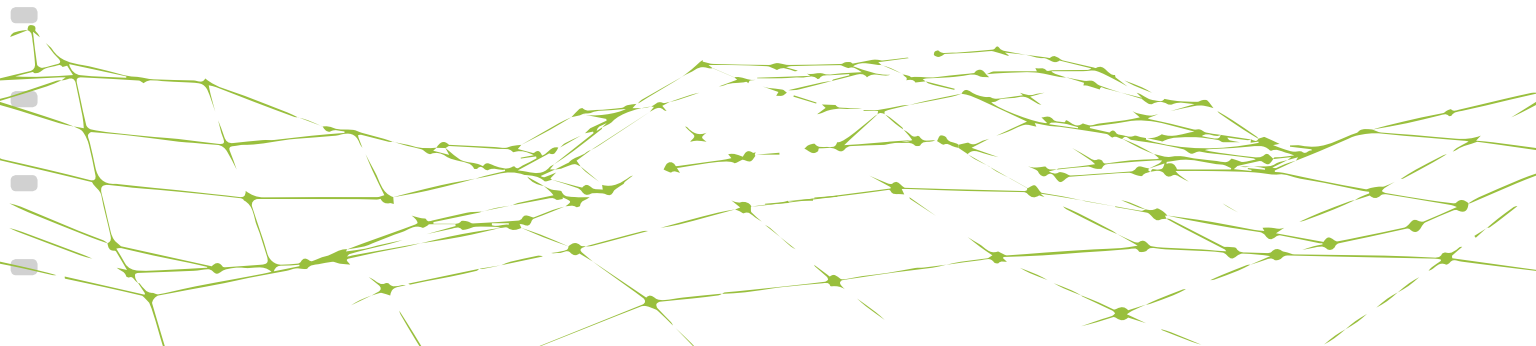
Whether you're new to the cloud, or you've been around since before cloud was cool, you are likely to see four common challenges that make the troubleshooting applications and infrastructure very difficult.



1. *Shifting Ownership*

The ability to roll your own architecture without the burden of dealing with physical devices is liberating and far more efficient. But as developer tools, deployment tools, and cloud operations tools become necessary bedfellows, the boundaries between development and operations become blurred or disappear altogether. This means the development team is suddenly an integral part of operations, whether by design or by default, adding complexity and distraction to developers' work.

The more time developers spend in the operations realm (troubleshooting applications or the cloud resources they depends on), the less time they have to do what they do best: create innovative applications and functionality.



2.

Lack of Transparency

In the past, the operations team was expected to prove that the hardware was working, the network was healthy, and the SAN hadn't lost disks before making the developer team dig in. As long as it was something "easy" in the infrastructure, the issues were manageable. When it was the application that had the issue, things got hard.

That pattern has been reversed with the cloud: now the burden of proof is on the developer team to find a problem that originated with someone else's complex, abstracted, virtualized data center.

& Burden of Proof

What if an application is returning an HTTP 500 error or performing poorly? If you're using something delivered as-a-Service, such as a database, queues, or cache, your visibility will be limited to the cloud provider's status page and whatever you can directly observe. It is either working correctly and performing well, or it isn't. Likewise, servers can be monitored, but you can't always tell why your virtual resource's performance has trailed off, especially if you are the victim of an environmental factor that is out of your control.

No matter how good the support team is at your favorite cloud provider, it's rare that they will be as responsive to your requests for more information on an issue as your own in-house ops team could be, and they won't be as well-versed on your architecture. To varying degrees, you are at the mercy of the cloud provider for consistent, reliable services. It is also up to them to offer timely insights when issues arise with the services you depend on. Cloud providers vary in the level of communication and transparency they offer. When insufficient, the burden of proof rests squarely with developers to show that the issue is not in the application.

3. *Easy Complexity*

Compounding the challenge of sorting through infrastructure issues vs. code issues is that applications are becoming far more complex, and in many cases, portions of the overall architecture may be transient in nature. Combine complexity with impermanence and you have a recipe for painful and mysterious issues.

The incredible thing about an SDDC is that you can easily create nearly any kind of architecture required to support your application stack's needs. If you can dream it, you can build it. Want to cobble together .NET, Java, PHP, Node.js, Ruby, Database-as-a-Service for SQL and NoSQL, Message-Queues-as-a-Service, and Search-as-a-Service?

From a cloud deployment perspective, it is easy to deploy and get started. But, with that ultra-polyglot approach and a heavy reliance on software-defined services comes a new set of challenges.

Developers deal with a variety of services that are, to them, essentially black boxes. Each of these services comes with its own set of tricks for gaining insight into performance and availability, but each one may differ in how to monitor and troubleshoot.

Learning how to support a variety of different technologies creates drag on your delivery velocity. It is difficult to learn performance and reliability tricks for a variety of technologies. As the learning curve is compounded, the focus shifts from building great software and making the business successful.

Not every monitoring tool can support every technology stack. The wider you cast the technology net, the harder it is to support your full stack from a single monitoring tool.

If you're using dynamic (transient) resources, such as scale-on-demand servers, you can lose critical data for troubleshooting a problem. It takes planning to preserve critical insights that disappear after a server is de-provisioned.

4. *Frequent Change*

The need to go faster is driving the processes and structural evolution around application development. With increased agility through the cloud, coupled with developer tools in the delivery cycle (think PaaS environments), timelines are being shorted, but releases are on the rise. This is especially true in organizations who have adopted agile development practices. Code can flow to production smoothly with greater frequency, and architecture changes can be made far more swiftly and easily. Unfortunately, with more frequent code releases and architecture changes comes more opportunities for something to break.

Agile and Lean promote always moving forward—rather than rolling back a release in the event of an issue. However, enabling these philosophies requires two things that are often missing in a slower moving environment, (1) the developer's visibility into a baseline of behavior telemetry, and (2) instant feedback on the health of the application post-release relative to the previously healthy baseline. Without this, it is hard to know if you have made gains or losses with your release. Your users are often your only real barometer.

So... How Do I Code More and Support Less?

There is no denying that the cloud has impacted the life of many developers, mostly in a very positive way. Of course, with new technologies and capabilities come a host of new challenges. In the case of cloud-hosted applications, this includes the ability to effectively and efficiently support those applications in their new environments. Otherwise, any gains in productivity are given back in support of the application.



There are 3 basic steps that every development team should take to make supporting cloud-based applications easier:

1. Establish Access, Process, and Protocol

For developers to support their cloud-based applications more effectively, they must have safe access to specific information and resources. Unfortunately, this is often an all-or-nothing proposition—full login rights to servers (and even management portal), or no access at all. It's important to establish appropriate access methods with developers so that they have the visibility they need, without handing over so much control that it increases the likelihood of accidents.

2. Design Supportability Into the Application

Once your application is in production, there are several common questions that you will need to answer at a moment's notice: Is it running? Are users satisfied with the performance? Is anything silently failing and frustrating users without setting off alarms? If something did fail, who was impacted, and what caused the issue?

There are also some things that simply cannot be measured and monitored from outside the application, but which can communicate it's health and well-being. To enable you to quickly answer the inevitable questions, consider incorporating the following:

- If it moves, measure it: Report application metrics and KPIs from within your code in order to see events and data that would otherwise be locked away from you. Some events and metrics can only be exposed by the developer. Knowing how your application behaves at a core level can provide invaluable insights when searching for troubleshooting clues. If you can configure monitoring and alerts for those metrics, even better.
- Log often, and log meaningfully: If you only report errors, you will lack the critical insights necessary to help point to the root cause of the issue(s). By logging at info or debug level, instead of just warn or error level, you will have the breadcrumb trail you need to find what's wrong. It is impossible to get the state of the system after the fact. You need to have logged it at the time of the event.
- Centralize your insights: Life in the cloud can be quite distributed, and quite transient. It's always good to bring everything—logs, errors, custom metrics, and other telemetry—into a central location for normalization, correlation, and continuity. You may need the data well beyond the ephemeral lifespan of your cloud resource.

3. Identify Health Baselines Early

Key information like message queue length, average request time, app pool resource utilization, custom metrics values, log and error rates, and more can be charted for your application. Monitoring and charting are not strictly for ops tools any longer. Understand what your application looks like when it is healthy and unhealthy, preferably starting with pre-production environments, so that you can see how your application morphs from release to release and with different loads as your architecture evolves. By baselining as far back as development and QA, you can often catch problems well before they impact customers and send you and your team scrambling.

Unravel an Application Troubleshooting Mystery

The cloud brings incredible capabilities to the lives of developers, like speed, agility, flexibility, and scalability. As with any new, disruptive technology, new challenges are expected. By applying some basic strategies for application management, monitoring and troubleshooting, you can have all of the advantages of the cloud without giving back the gains on critical support issues. Needless to say, the results are happier and more productive teams.



Application
Monitoring vs.
Application
Performance
Management
(APM)

Cloud based applications are highly dynamic and can span a variety of servers and services. Supporting them requires access and correlating details across multiple sources and tools. Agile development makes release cycles shorter, forcing developers to be more involved in day-to-day IT operations. At the same time, it reduces developers' ability to get the information necessary to resolve an application's performance issues, creating a need for application monitoring and application performance management.

There are a lot of tools on the market that provide some of the information, but when an issue occurs it is difficult to pinpoint the problem across multiple tools, servers, and environments. Below is a list of various tools and the role they can play in troubleshooting application problems.

Log Aggregation/ Log Management

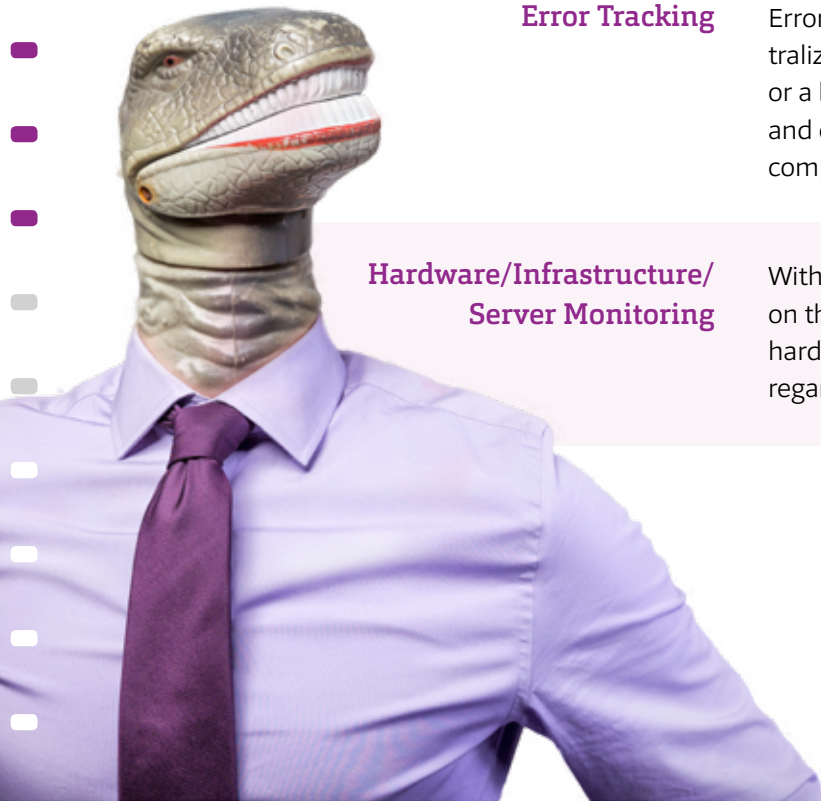
These tools typically allow you to aggregate log files from different servers, environments, and applications for easy access to a centralized repository for files. This allows easier search, alerting, and notifications. As more companies move to the cloud and require scale up and down mechanisms, having a centralized log repository becomes more important. A server that was active when an issue occurred may not be active when troubleshooting. Without having a log management system, the data could be lost forever.

Error Tracking

Error or exception tracking solutions typically allow for centralized error tracking so that every occurrence of an error or a bug is presented. This allows developers to see trends and error rates, browse and search through the data, and get comprehensive error detail information.

Hardware/Infrastructure/ Server Monitoring

Within this category you'll find various solutions. Some focus on the network and some on the server, but they all collect hardware related data and provide visibility and notifications regarding the status of the application infrastructure.



Application Monitoring vs. Application Performance Management (APM)

These are two different things, though some companies use the terms interchangeably. As Wikipedia describes it, “Since the first half of 2013, APM has entered into a period of intense competition of technology and strategy with a multiplicity of vendors and viewpoints. This has caused an upheaval in the marketplace with vendors from unrelated backgrounds (including network monitoring, systems management, application instrumentation, and web performance monitoring) to adopt messaging around APM. As a result, the term APM has become diluted and has evolved into a concept for managing application performance across many diverse computing platforms, rather than a single market.”

Application Monitoring

According to Technopedia, application monitoring is “a process that ensures that a software application processes and performs in an expected manner and scope. This technique routinely identifies, measures, and evaluates the performance of an application and provides the means to isolate and rectify any abnormalities or shortcomings.” What it does not do is talk about how deep the monitoring goes. We define it as basic application monitoring which gives high level indications of the application’s health and behavior by measuring various element’s response times.

Application Performance Management (APM)

APM takes application monitoring to a deeper level, in which the application is analyzed down to the code-level to provide indications of functionality, as well as specific elements in the code that do not perform as designed and require a fix or optimization.



3 Data Sets That Bring Context to Exceptions

Even the smallest exception can cause material damage to your company's operations, which in turn can affect revenue. It is important to have a plan to track and troubleshoot exceptions. It seems like a simple process—the exception is raised, you look at it, and then make changes to your application so it never happens again. But we all know it's never that easy.

Often, an exception only indicates that something has gone wrong. It will not tell you why something went wrong, or how the issue occurred. This does little to simplify a developer's ability to fix the problem.

You cannot rely on a single exception because it lacks context. In the case of troubleshooting an exception, context allows you to fully assess and understand the cause of an exception.

Following are three basic data sets (and there are more) that can help provide context.



1. Other Exceptions

You may already know what exception started the troubleshooting process, but you gain an understanding of the severity and intensity of an exception by determining how frequently it occurs, specifically when it occurs, and what other exceptions occurred around the same time. Seeing exceptions across application boundaries can be especially powerful in modern distributed systems. If your web application throws an exception that is triggered by a web service it is consuming, you need to follow the trail to the web service and correlate the exceptions originating from the web service. In some cases, you may be correlating exceptions across multiple applications, servers, data centers, and even across different languages. It is not always going to be easy to obtain exception data, so it's important to work closely with IT to set up necessary processes and tools.

2. Log Data

If you log extensively, your application logs have the potential to be the exposition of every method, process, and task taken by your application leading up to the exception. They paint the step-by-step picture of both the how and the why. Getting log data can be equally challenging, whether IT owns the servers, logs live on by virtue of running your applications in a dynamic cloud environment. In the cloud, servers come and go more frequently, making it harder to get the logs even if you solve the IT access issue. When it comes to multiple servers running many applications, the amount of log data needed to make sense of an issue increases exponentially. Finding the relevant pieces quickly and connecting the dots between exceptions and the relevant log statements is key.

3. Application and Environment Health Metrics

An exception can often occur as a result of something infrastructure-related: database timeouts, full disk, out of memory, network issues, etc. Sometimes the root cause is obvious from the exception itself, but other times you need to correlate the error with the application and environment state to see where the aberrant behavior was at that exact moment. Furthermore, if you are experiencing sporadic problems with your environment or a third-party resource, combining error telemetry and application environment health metrics can point out trends and the impact of any problems. This is extremely useful when the burden of proof is on the developer to get a cloud partner to fix an issue on their end.

While some exceptions are easy enough to address with little or no context, it is far more likely you will need additional data. This can take you from information about what happened, to having key insights that explain why.



Rapid Dev Improvement

Developers & the Full Application Lifecycle

The most common reaction we get to our BuildBetter eMag is, "what is this?" True we aren't a publishing company, but we are a developer company, and that's bigger than building software.

Applications are taking over the world, and the role of the developer is moving from the back room to the forefront of any business. To meet rapidly increasing demands, developers need to focus their time on what they do best: building amazing solutions. These are not only cool ideas, but tools that solve real problems as designed.

Stackify exists to increase developers' ability to create amazing applications as quickly as possible. We believe that the ideal development team, today and in the future, is consistently optimizing its output across the entire lifecycle of an application; from development to testing to production. Stackify's mission is to give

developer teams easy access to powerful tools, which enable them to take the lead in delivering the best applications as quickly as possible.

For more information on our developer tools, visit stackify.com.

For more dev-centric information and resources, visit our content hub, [BuildBetter Online](#).

And as always, we love to hear your comments and requests: mhobbs@stackify.com

Build on,



Max Hobbs

Stackify CMO & BuildBetter Co-Editor

Dial Up Your Next User Group Meeting!



[Request the Stackify DevBox](#)

Call for Entries:

BuildBetter is both an eMag and a site hub. We're always open to content from you, like:

- Topic suggestions/general ideas
- Blog articles
- Interview subjects (volunteers, nominations)

[Please contribute!](#)

[Submit to BuildBetter Team](#)