Realtime publishers

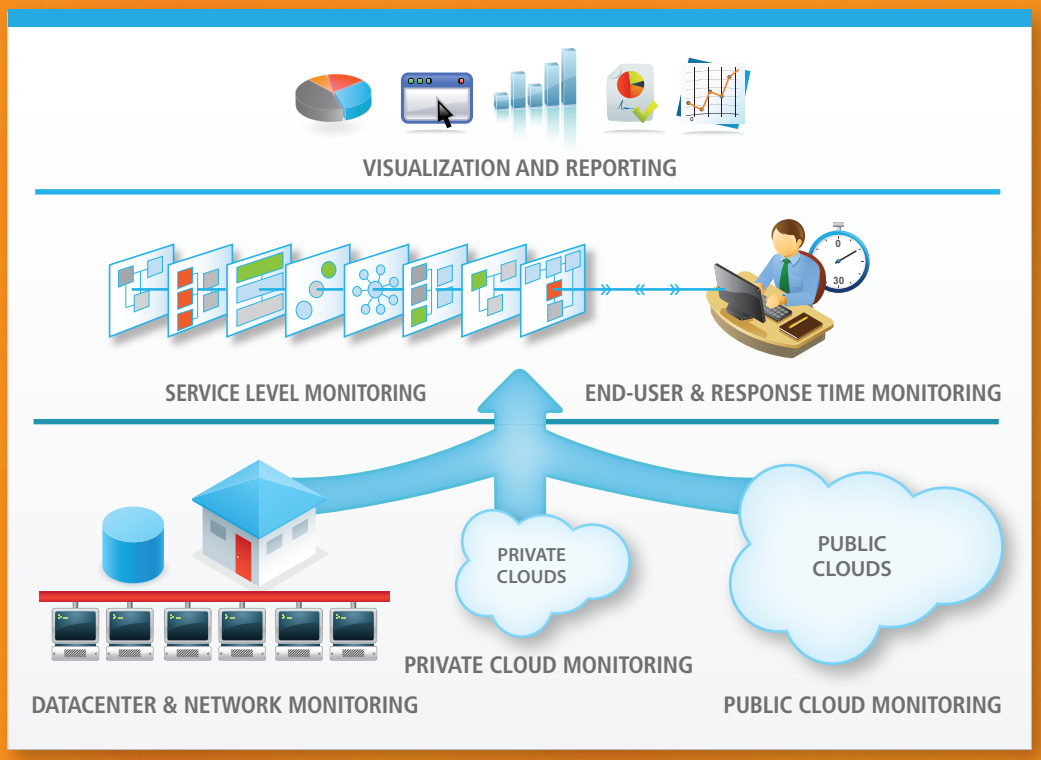
The Definitive Guide™ To

Monitoring the Data Center, Virtual Environments, and the Cloud

sponsored by

Don Jones

The Nimsoft Monitoring Solution



- **Ensures business service delivery regardless of IT platform**
- **Enables rapid adoption of new computer infrastructure such as private and public cloud**
- **Monitors the datacenter to the cloud, including SaaS, hosted, and virtualized environments**
- **Lowers TCO by 80% and delivers proven value in weeks**

Unified Monitoring

nimsoft
From the Datacenter to the Cloud

www.nimsoft.com

Copyright Statement

Chapter 2: Traditional IT Monitoring, and Why It No Longer Works

Those of us in the IT world *think* we know monitoring. After all, we've been doing it, in various ways and using various tools, for decades. We collect performance data, we look at charts, and…well, that's monitoring. Sadly, that kind of monitoring just doesn't meet today's business needs.

How You're Probably Monitoring Today

IT monitoring has evolved over the past few decades, but that evolution has pretty much consisted of continuing refinements to a basic model. Today's monitoring techniques evolved more out of what was *possible* and less out of what the business actually *needed*. Let's take some time to look at the monitoring techniques of today, because we'll want to carefully consider which techniques we need to keep—and which ones we should ditch.

Standalone Technology-Specific Tools

Today, you're probably relying heavily on monitoring tools that are standalone and technology-specific. That is, once you move beyond the collection of basic performance data, you start to move into extremely domain-specific tools that are geared for a particular task. You might, for example, use a tool like SQL Profiler (see Figure 2.1) to capture diagnostic information from a Microsoft SQL Server, or you might use a tool like Network Monitor (see Figure 2.2) to capture network packet information.

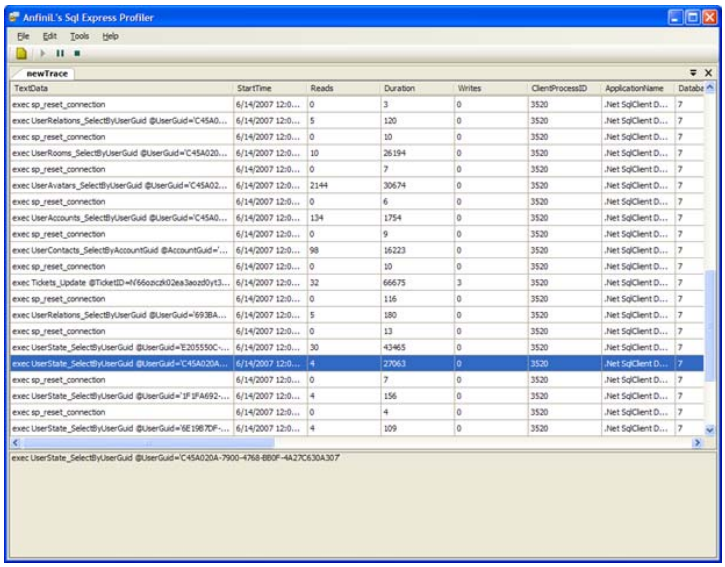


Figure 2.1: SQL Profiler.

The problem with these tools is that they *are* domain-specific, and they require a great deal of domain-specific knowledge. You have to know what you're looking at. Although these tools will *always* provide us with valuable troubleshooting information, they don't tell us much about the health of an application that runs across multiple technology domains. In fact, in some instances, these domain-specific tools can lead to longer and more convoluted troubleshooting processes.

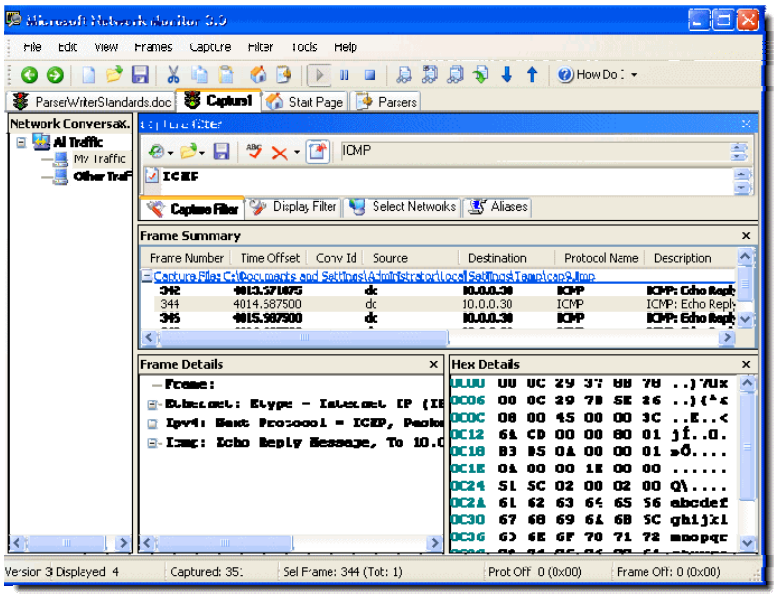


Figure 2.2: Network Monitor.

For example, when a user complains of a slow application, a database administrator might grab SQL Profiler to see what's hitting the database server. At the same time, a network administrator might start tracing packets in Network Monitor to see if the network looks healthy. Both of them are failing to see the forest for their individual trees, and failing to recognize that the application's health isn't driven entirely by one or the other technology domain.

Local Visibility

Our current monitoring tools are, quite understandably, limited to our local environment. We monitor *our* servers, *our* network, *our* infrastructure components, *our* software applications. The minute we leave our last firewall, we start to lose our ability to accurately measure and monitor; at best, we can get some response time statistics from routers and so forth on our Internet service provider (ISP) network, but once we're out of our own environment, our vision becomes severely limited.

Technology Focus, Not User Focus

Even tools that profess to monitor an entire application "stack" still take a very domain-centric approach. For example, it's not uncommon to have tools that continuously collect performance information from individual servers and network components, compare that performance to pre-determined thresholds, and then display any problems. These tools can be configured to understand which components support a given application, so they can report a problem that is affecting the application's health and help you trace to the root cause of that problem. Figure 2.3 shows an example of how these solutions often present that kind of problem to an administrator.

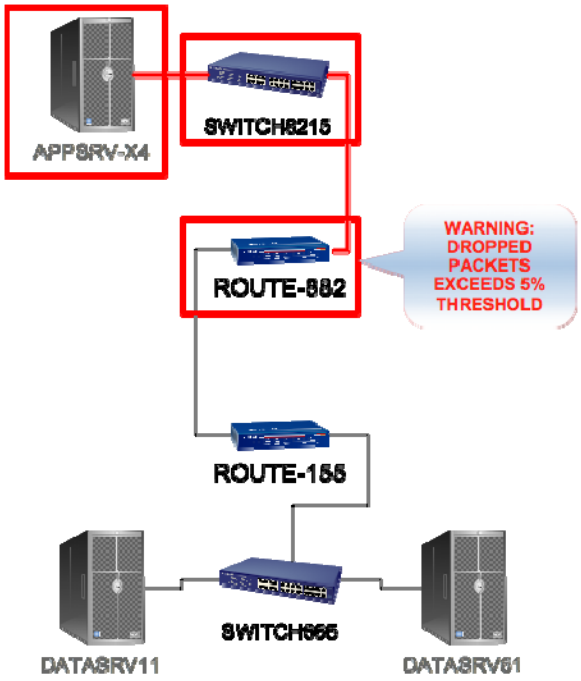


Figure 2.3: Tracing application problems to a specific component.

These tools, however, don't encourage a user-centric view of the application; they encourage a technology-centric view. They concern themselves with the health and performance of application components, not with the way the end user is currently experiencing that application. These kinds of tools can absolutely be valuable, but only when they can also include the end users' experience at the very top of the application's stack, and when they can do a better job of correlating observed application performance to specific component health or performance.

Problems with Traditional Monitoring Techniques

In addition to the problems I pointed out already, our traditional monitoring techniques have some severe shortcomings that actually make monitoring and application health maintenance more difficult than it should be.

Too Many Tools

For starters, we simply have too many tools. They deliver too much different information in too many different ways. There's no way to correlate information between them, and we have to spend a ton of time becoming an expert on every single tool's nuances and tricks. Consider a modern, multi-tier application: It might rely on several servers, a database, network connectivity, and so on. When the application "seems slow" to the users, you have to reach for a dozen tools to troubleshoot each element of the application stack—and you're still not looking at the application as a whole.

We need to get centralized visibility of the entire application and all its components. We need information presented in a uniform, consistent fashion, and we need it correlated so that we can tell which bit of the application stack is contributing to an observed problem with overall application health.

Fragmented Visibility into Deep Application Stacks

Another problem with our traditional monitoring tools is that they're really not built for today's deep application stacks. Consider what might seem like a fairly straightforward multi-tier application, consisting of:

- Client application
- Middle-tier application server
- Back-end database server

That doesn't include the connectivity components, though, so let's include them:

- Client application
- Network switch
- Network router
- Network switch
- Middle-tier application server
- Network switch
- Back-end database server

Each of those individual elements, however, is really a stack unto itself:

- Client application
    - Java runtime engine
    - Java libraries
    - Operating system (OS)
- Network switch
- Network router
- Network switch
- Middle-tier application server
    - .NET Framework runtime engine
    - .NET Framework classes
    - Database drivers
    - Operating system
    - Memory
    - Processor
- Network switch
- Back-end database server
    - OS
    - Database management system
    - Disk subsystem
    - Memory
    - Processor

All these sub-components can have a significant impact on application health, but it's very difficult to get traditional monitoring tools that can "see inside" all of them. We might be able to get excellent data on the database management system's performance and use of memory and processor resources, while having virtually no idea how the middle-tier application's database drivers are performing. This fragmented visibility makes it tough to find the root cause of problems.

Disjointed Troubleshooting Efforts

Domain-specific tools lead to domain-specific troubleshooting. Let's revisit my case study illustration from the previous chapter and see how this domain-specific troubleshooting usually works in the real world:

> John, the IT specialist at World Coffee, is trying to find the cause of performance problems that Ernesto has reported in the company's order management application.
>
> John initially suspected that the database server was running slowly, and he notified the company's DBA. The DBA, however, said that the individual queries are executing within the expected amount of time, and that the database server's overall performance looks good. John then called a desktop support technician to look at Ernesto's computer. The technician said that everything on the computer seems to be running smoothly—the problem seems isolated to this one application. A software developer that John contacted insists that the application is running fine on his computer. John is now analyzing network packet captures to see whether there's some latency between the server network segment and the client segment that Ernesto's computer is connected to.

Sound familiar? This is how most companies deal with application health issues today: A bunch of domain experts jump on their particular application component, tending to look at that component in isolation and tossing the problem "over the wall" to another specialist when they can't find an obvious problem with their particular component.

Application stack tools like the one shown in Figure 2.3 are a good starting point for solving this problem because they help pinpoint the component that isn't performing to specification. But they fail in that they're still looking at each component as a standalone entity, and measuring performance against predetermined thresholds. It is entirely possible for a server to be within its performance tolerances and yet *still* be the root cause for a poorly-performing application; these tools don't go far enough in that they don't correlate observed application behavior with component performance.

Difficulty Defining User-Focused SLAs

We don't tend to offer user-centric service level agreements (SLAs) because our monitoring tools don't really let us figure out what "good" end user experiences should look like. We can tell you that the database server running at 90% processor utilization isn't good, but we can't tell you exactly how that will manifest in end user experience. Simply put, we're too focused on the technology and the components and *not* on the application and its end users.

As we start moving into hybrid IT and outsourcing some of our applications, we need to focus less on the technology—which isn't going to be in our control anyway—and focus instead on getting what we're paying for, which means focusing on the service that our end users are receiving.

No Budget Perspective

Finally, another growing problem with traditional monitoring tools is that they really don't have any budgetary focus. That's not necessarily a huge deal for in-house applications, but as we start moving into hybrid IT and outsourcing portions—or all—of an application, we need to know that we're getting what we paid for. When we start moving to pay-as-you-go cloud computing, we need tools that will help correlate application health *and use* to that pay-as-you-go model so that we can accurately forecast and plan for those cloud computing expenses.

Evolving Your Monitoring Focus

IT is rapidly evolving toward hybridization; every day, companies adopt Software as a Service (SaaS) solutions, outsource specific services to Managed Service Providers (MSPs), and move applications and components into cloud computing platforms. As IT evolves, so must our ability to monitor these assets to ensure they are performing to our needs.

The End User Experience

The first point of evolution is to focus entirely on the end-user experience (EUE) as your top-level metric. The first and foremost thing you should care about is how quickly your end users are able to perform selected tasks with an application.

When a problem occurs in an application—whether it's something in your control, like your local network, or something outside of your control, like a back-end database server in a service provider's data center—that problem will "flow up" through the application stack, resulting in a problem with the EUE. *That* should be your indication that there's a problem: When the end user *experiences* the problem.

Does every application problem impact the EUE? No, of course not. A service provider might lose a server but might also have redundancy built-in to handle that exact situation. If you don't see a problem in the EUE, *you* don't have a problem. With the right tools, you'll be able to start at that EUE and drill down to find the root cause of problems that *are* under your control, making the EUE the perfect place to begin problem diagnoses and troubleshooting activities.

Figure 2.4 shows how a monitoring solution can expose that EUE in a simple fashion, such as through a color-coded "response time" indicator, as shown. Green means the end users are going to have an acceptable experience; anything else requires your attention.

Figure 2.4: Top-level monitoring of the EUE.

The Budget Angle

As you move into pay-as-you-go services, you'll want your monitoring to be correlated to your expenditure. Bringing all that information together into a single console, such as the one illustrated in Figure 2.5, can help you predict expenses and plot growth in your service consumption and the associated expenses.

Net In	Bytes
Average	1694407.00
Sum	8472036.00
Minimum	520848.00
Maximum	3938920.00

Net Out	Bytes
Average	2077689.00
Sum	10388445.00
Minimum	549269.00
Maximum	5237538.00

Disk Write	Bytes
Average	0.00
Sum	0.00
Minimum	0.00
Maximum	0.00

Disk Read	Bytes
Average	0.00
Sum	0.00
Minimum	0.00
Maximum	0.00

CPU Usage	%
Average	14.00
Sum	71.00
Minimum	5.00
Maximum	38.00

Figure 2.5: Monitoring cloud computing consumption.

Traditional Monitoring: Inappropriate for Hybrid IT

If you've been following my logic closely to this point, you may be ready to make a significant argument: *Traditional monitoring can do all of this.*

True. To a point. We already do have tools that let us measure things like service response times, and many companies use those to develop a top-level view of application health—almost a sort of EUE metric. That works fine when you're *completely inside your own network,* but the minute you start creating a hybrid IT environment, you lose whatever deep monitoring ability you may have. Understand, too, that hybrid IT *doesn't* just mean that you've outsourced a few services. It means that you may have internal services that *depend on* external services. For example, consider Figure 2.6.

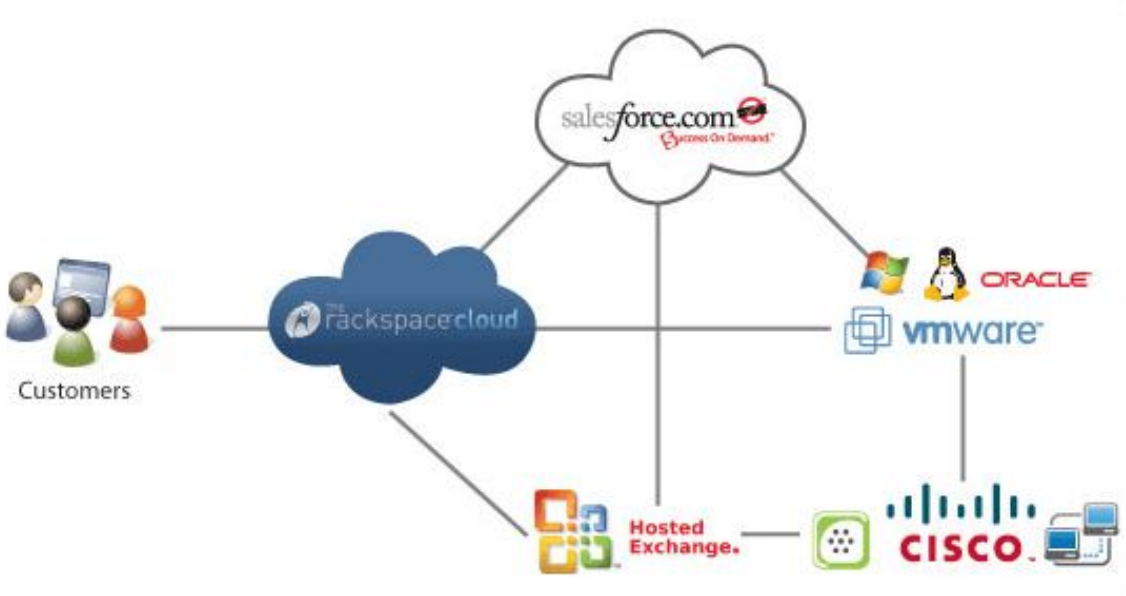


Figure 2.6: A truly hybridized IT environment.

Illustrated here is an e-commerce application, hosted in the Rackspace Cloud platform. That application requires access to SalesForce.com, a SaaS customer relationship management (CRM) solution. It also depends on an Exchange Server messaging system, which is hosted by an MSP. Finally, it relies on data from within your own data center, perhaps running on Windows or Linux computers, which might in turn be virtual machines that depend on a virtualization platform such as VMware. Your traditional IT monitoring tools simply can't put all those varied components into a single view for you. And how will you monitor the EUE? Ask your customers to install a little monitoring agent on their computers? Probably not—that's usually known as "spyware" even if it's being used for noble purposes.

When you start moving to complex, hybrid IT environments, the old let's-get-a-bunch-of-tools approach just doesn't work anymore. There's another problem, too, that can't be corrected using traditional monitoring tools: The problem of correlating performance to your real-world business.

It's Your Business, So It's Your Problem

Let's be very clear on one thing: Hybrid IT involves outsourcing *business services;* it does not entail handing off responsibility for your business. And hybridization is not a panacea for all your IT woes; although it can solve many business problems, it can also introduce unique challenges.

Provider SLAs Aren't a Business Insurance Policy

Consider this troubling possibility:

> John is having a bad day. Last month, World Coffee moved one of its smaller applications to New Earth's cloud computing platform, and for most of today, that application has been completely unavailable. John's boss—and his boss, and his boss—is furious; the company estimates that it's losing $250,000 every hour that the application is unavailable. Much of that is in lost sales, and the company will not only have difficulty recouping the money but also may lose some of those customers to other distributors. So far, they've lost more than a million and a half dollars.
>
> Just then, the application comes back online. Relieved, John calls his contact at New Earth to follow-up. He explains to his contact that the company has lost hundreds of thousands of dollars in sales, and that he wants to invoke New Earth's SLA terms.
>
> His contact agrees but regretfully informs John that World Coffee isn't due any money from New Earth. The SLA provides for a refund of fees paid to New Earth for any services that were not provided in accordance with the SLA. However, World Coffee is on an entirely pay-as-you-go plan, meaning they have no fees beyond those they pay for what they actually used. Since they haven't been using the application for the past 6 hours, they would have paid New Earth nothing, and so New Earth can't offer any refund.

That's how most SLAs work: They're there to guarantee the service or your money back (or a portion of it). They're *not* there to cover the business losses that an outage or performance problem may cause. For example, Microsoft's Windows Azure Compute SLA states that customers can receive a 10% service credit when monthly uptime percentage falls below 99.95%, or a 25% credit if uptime is below 99%. The SLA doesn't provide any performance guarantees; if the service is up but responding slowly, customers aren't due any credit.

This is exactly why it's so crucial that we be able to monitor performance across our hybridized IT infrastructure. If we're not seeing the EUE that we need, we can take action, either by working with service providers to raise performance or finding new service providers.

Concerns with Pay-As-You-Go in the Cloud

Many cloud computing platforms operate on a pay-as-you-go model, which is one of the main points that make them so attractive to businesses. They offer the ability to scale to almost infinite resources, provided you're willing to pay for that. When you don't need a lot of resources, you're not paying much. So you can get a giant potential infrastructure with essentially zero capital investment.

But pay-as-you-go can be full of surprises. Ever go shopping for songs on Apple's iTunes store? Each song is only a dollar or so, so you click, click, click—and then a few days later get a bill for a couple of *hundred* dollars. Oops. These things add up, don't they?

Cloud computing can be the same way. To continue picking on Windows Azure as an example, pricing at the time of this writing is as follows:

- Computing power costs between $0.12 and $0.96 per compute-hour, depending on the size of the compute instance.
- Storage costs $0.15 per month per gigabyte, and $0.01 per 10,000 storage transactions—meaning data reads and writes.
- Data transfers cost $0.10 to $0.45 per gigabyte, depending on the direction (in or out) and the region of the world.

Sounds cheap—pennies for gigabytes! But how much will your application use? This is where the performance=budget angle comes in. You don't want to get that surprise bill at the end of the month; you want to be able to monitor your use of the application and *predict* what your bills will be, and even use trending to predict changes in usage patterns and the resulting bills.

Evolving Monitoring for Hybrid IT

So let's talk about what kind of monitoring our hybrid IT environment is going to need. I'll refer to this as *evolved monitoring,* to draw a distinction between this new approach and the more traditional techniques you're already using. Hybrid IT is an evolution in how we deploy IT services, so it only makes sense that we'd need some evolved monitoring capacity to go with it.

Focusing on the EUE

The first thing we need is a monitoring solution that focuses on the EUE—it's the first and foremost metric that we should care about, and our tools should focus on what *we* care about. A tool should be able to help us test the EUE of an application, either by passively observing our application in action or by actively "probing" our application to measure the results. As Figure 2.7 shows, the EUE should be broken down into the various contributing components so that we can see how each component drives the overall EUE.

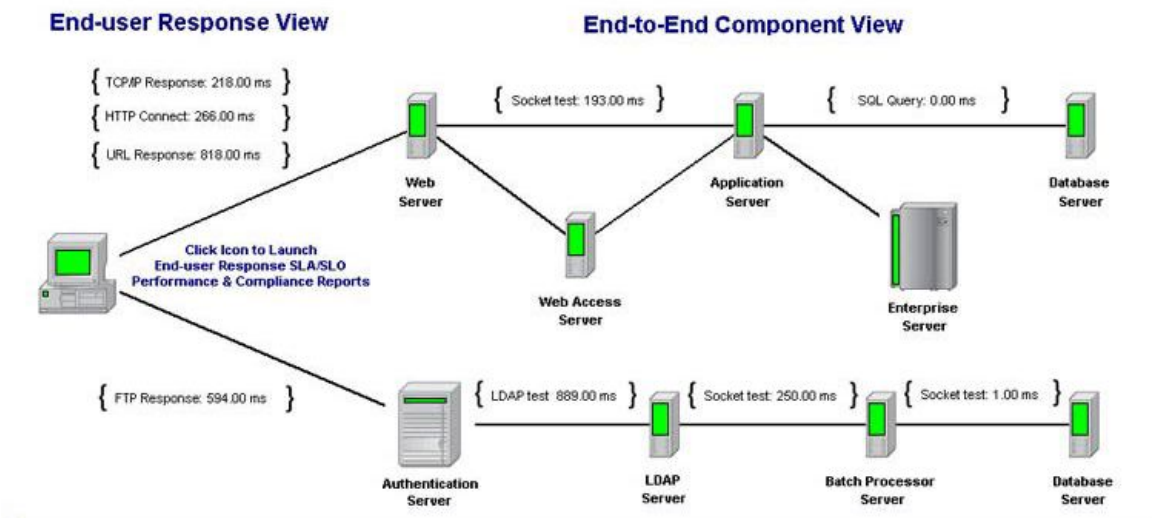
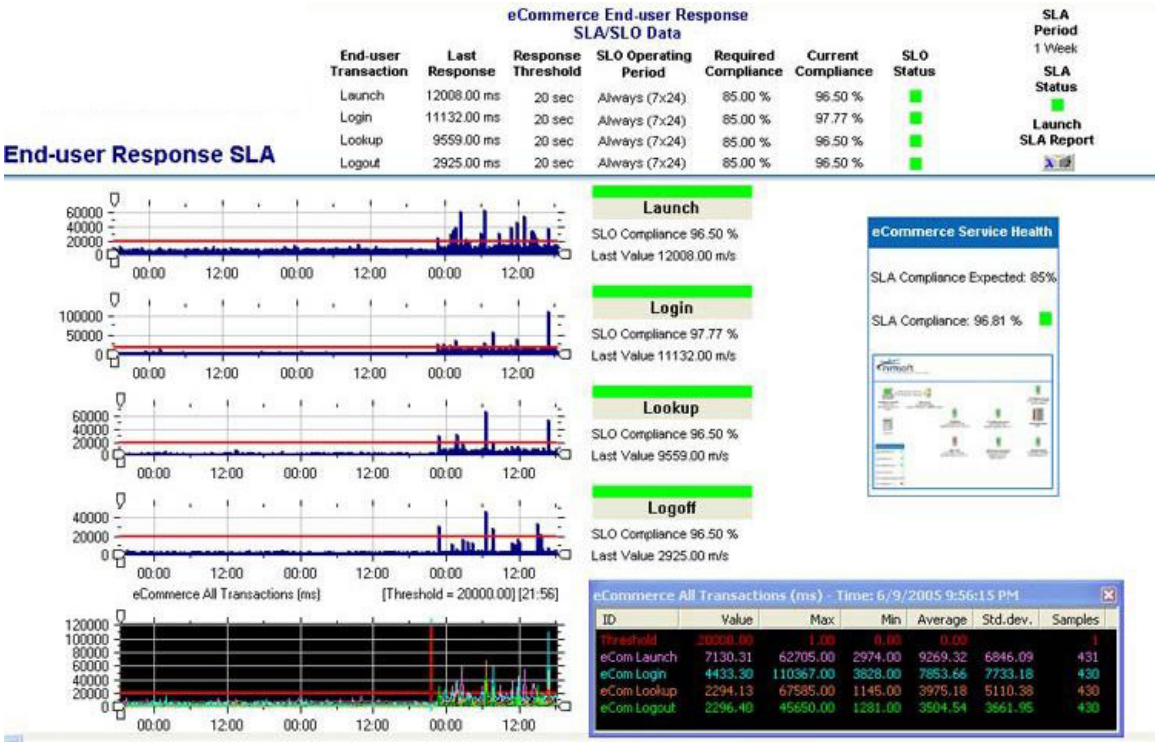


Figure 2.7: Breaking down the EUE.

This approach not only helps us manage to that all-important EUE metric but also provides a starting point for diagnosing problems when that metric strays out of our comfort zone. Our tool should allow us to define our EUE-based SLAs, and should help us monitor compliance with those SLAs. Figure 2.8 shows what that might look like.



End-user Transaction	Last Response	Response Threshold	SLO Operating Period	Required Compliance	Current Compliance	SLO Status
Launch	12008.00 ms	20 sec	Always (7x24)	85.00 %	96.50 %	
Login	11132.00 ms	20 sec	Always (7x24)	85.00 %	97.77 %	
Lookup	9559.00 ms	20 sec	Always (7x24)	85.00 %	96.50 %	
Logout	2925.00 ms	20 sec	Always (7x24)	85.00 %	96.50 %	

ID	Value	Max	Min	Average	Std.dev.	Samples
Threshold	20000.00	1.00	0.00	0.00		1
eCom Launch	7130.31	62705.00	2974.00	9269.32	6846.09	431
eCom Login	4433.30	110367.00	3828.00	7853.66	7733.18	430
eCom Lookup	2294.13	67585.00	1145.00	3975.18	5110.38	430
eCom Logout	2296.40	45650.00	1281.00	3504.54	3661.95	430

Figure 2.8: Managing end user response SLAs.

At the top of this console, you can see that specific end user transactions, like logging in and logging out, have been defined with SLAs, and the console is monitoring those transactions and telling us how often the application fails to meet our defined response times. We should even, as Figure 2.9 shows, be able to pull up a history of SLA compliance.

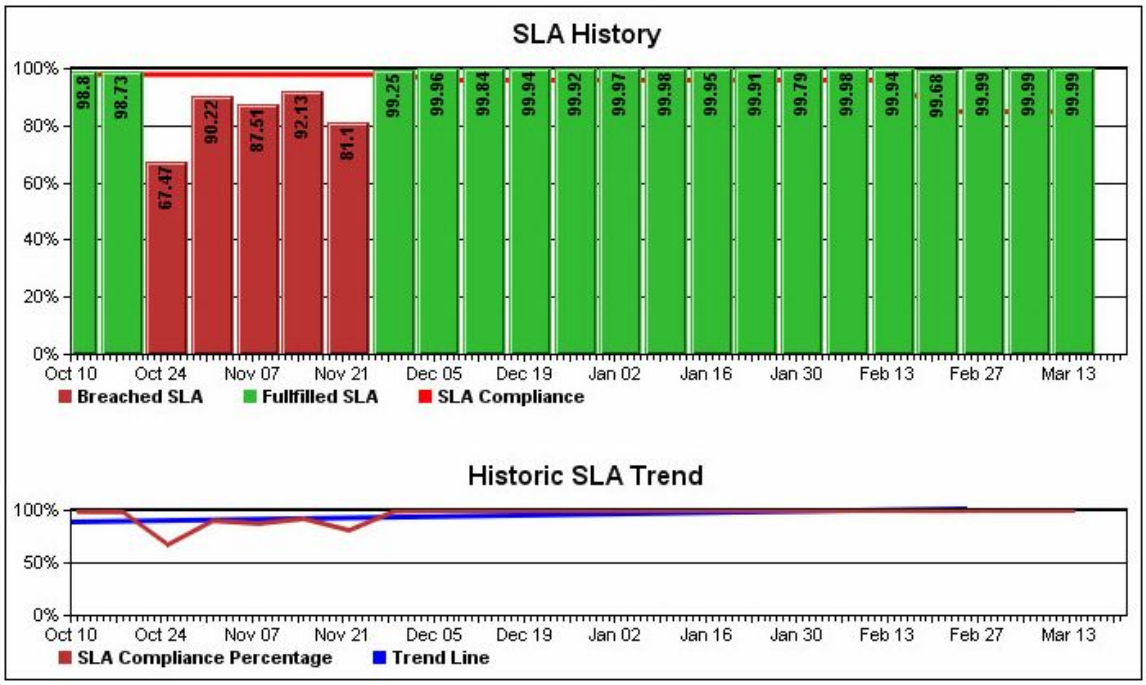


Figure 2.9: Monitoring SLA compliance over time.

Note that these SLAs are being defined by simple response times like pinging a service; they're times to *complete a specific end-user task*. That's a distinction we'll drill into in the next chapter.

Monitoring the Application Stack

Being aware of the EUE metric is important; however, when it's outside of our acceptable range, we need to be able to drill further to find a problem. That's why an evolved monitoring solution needs to be able to drill deeper, measuring specific application components. Figure 2.7 provided one example of how that might be visualized in a monitoring application; Figure 2.10 shows another.

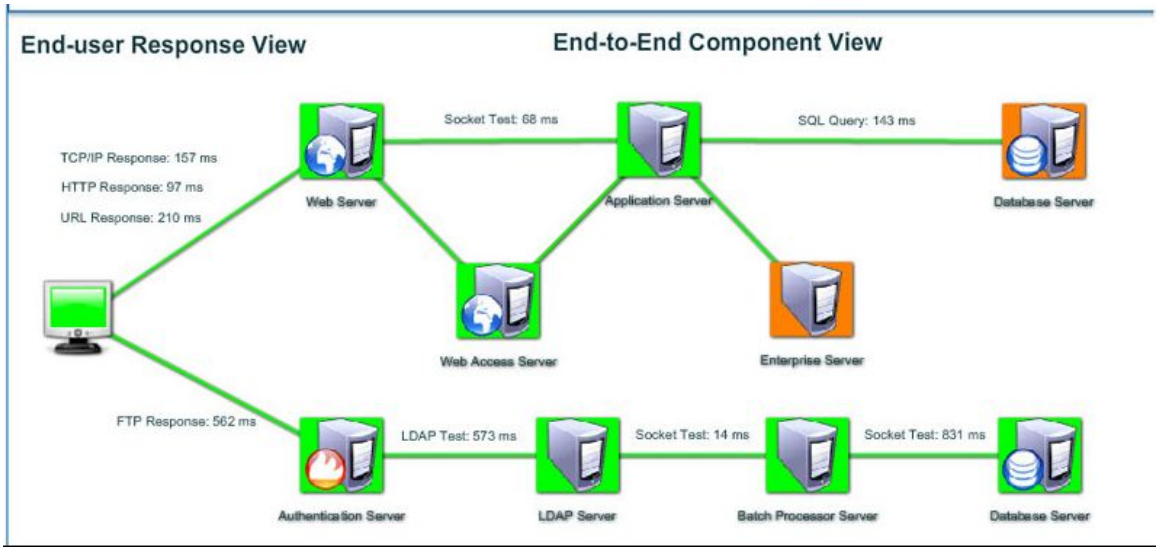


Figure 2.10: Monitoring the entire application stack.

From this kind of view, you should be able to drill deeper into specific problem areas. That drill-down should help you spot problems with *that particular* component. For example, clicking on "Enterprise Server" in the end-to-end view might bring up a drill-down console like the one Figure 2.11 shows, where we can get a high-level view of that particular server's performance, and start looking for problems.

Figure 2.11: Drilling down into a server's performance.

This drill-down must be aware of the kind of component we're looking at. For example, this view might be sufficient for a typical Windows server, but it's not showing us anything specific for a database server. If we'd clicked on a database server, we'd expect to see information related to the database management platform, such as the console shown in Figure 2.12.

Figure 2.12: Drilling down into a database server.

Now, we're really troubleshooting problems. We can see that the server's CPU is dangerously close to maximum, and database response times have crept firmly into the red. We're running low on free space, too. With a couple of clicks, this all-in-one console might take us from a potentially-problematic EUE metric right to the root cause of the problem—which we can now confidently assign to a DBA to start fixing.

Keeping an Eye on the Budget

Because it's continually monitoring our application, this evolved console can also help us keep an eye on our expenses for pay-as-you-go services. By simply tracking the things we pay for—compute time, bandwidth, storage, and so on—we can use the console to help plan growth and learn what to expect in our bills. We might even be able to export those numbers into a spreadsheet and play "what if" scenarios to see what our bills would be like if we increased or decreased our application activity.

Coming Up Next…

We need to redefine what "service level" means, and to do that, we need to think about what really matters in IT: the EUE. In the next chapter, we'll go into detail about why that EUE is so important from a business perspective, and why that must drive the technology perspective. We'll also look at the unique challenges imposed by a hybrid IT environment, and how that kind of environment can make it even more challenging to accurately assess the EUE. We'll examine the technologies and techniques available for monitoring the users' experience, and some of the reasons why that kind of monitoring isn't already in your arsenal of tools. We'll also look at some of the unique perspectives that service providers have on the EUE, and how they can help themselves and their customers do a better job of maintaining application performance.

Download Additional Books from Realtime Nexus!

Realtime Nexus—The Digital Library provides world-class expert resources that IT professionals depend on to learn about the newest technologies. If you found this book to be informative, we encourage you to download more of our industry-leading technology books and video guides at Realtime Nexus. Please visit http://nexus.realtimepublishers.com.