Realtime publishers

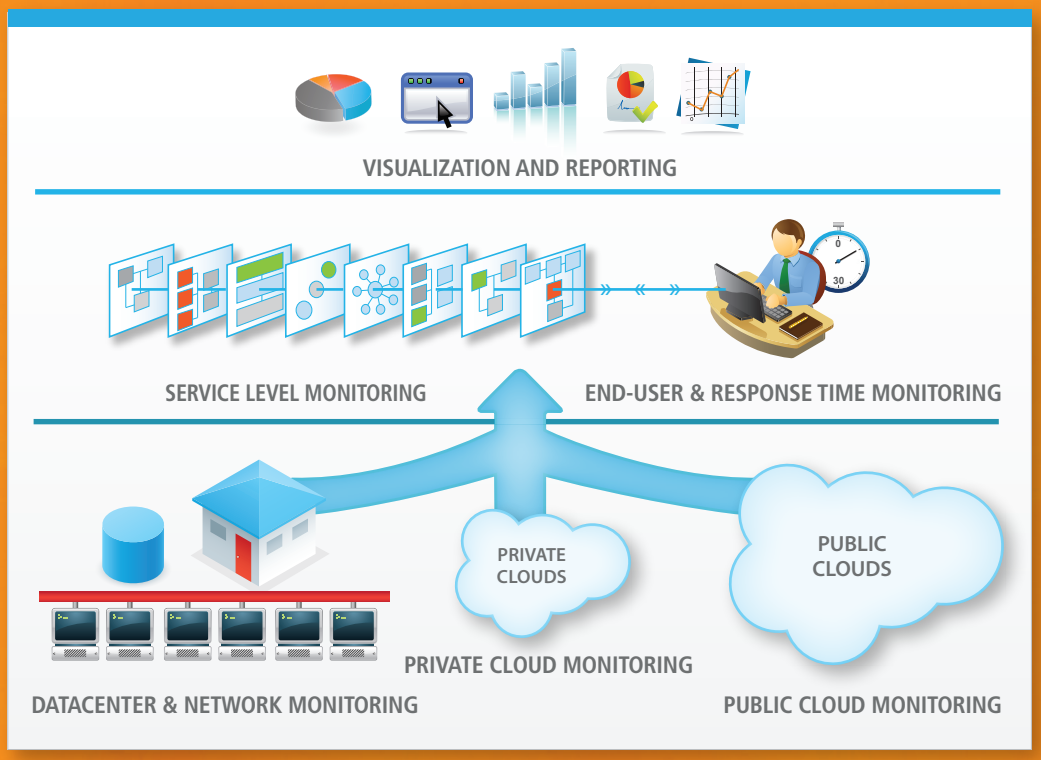
The Definitive Guide™ To

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 4: Success Is in the Details: Monitoring at the Component Level

The EUE is your ultimate metric for whether an application is performing well. It's what you should base SLAs upon, and it's certainly your ultimate measure of success or failure with an application. The EUE isn't, however, very useful at helping you *troubleshoot* problems when they occur. For that, you'll need a deeper, more detailed level of monitoring. In this chapter, I want to compare and contrast two approaches for that more-detailed kind of monitoring: the traditional, multi-tool monitoring stack, and a more modern approach that focuses on getting everything into a single view.

Traditional, Multi-Tool Monitoring

In a traditional monitoring environment, IT experts tend to rely on single-discipline tools to troubleshoot the application stack. That's an approach that has served for years, although as applications become more complex and distributed, we've needed a wider number and variety of tools to get insight into everything. Typically, separate tools exist for each major layer of the application. Consider the example in Figure 4.1, which illustrates the application stack I'll be using for this section of this chapter.

This diagram is more hardware-centric, as it is intended to represent the major physical elements of the application: client application, network infrastructure, application server (which might be a Web server, for example), and a back-end database. Notice that this example—in keeping with the "traditional" approach—does not incorporate any cloud-based elements; we'll come to the cloud issue shortly, though.

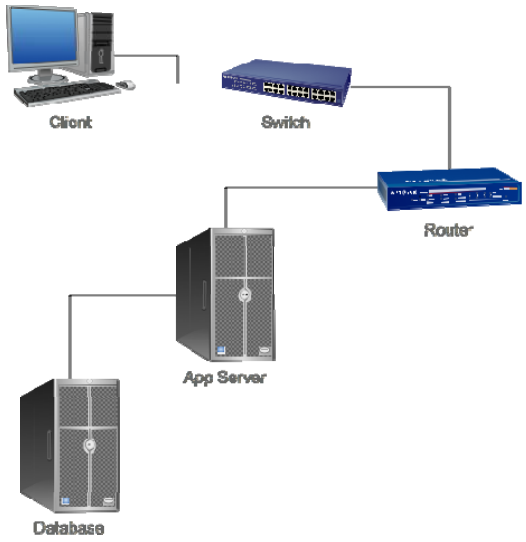


Figure 4.1: An example application stack.

Let's look at each of these elements in turn.

Client Layer

The client obviously plays a major role in an application's performance. A slow client computer can do more to ruin the perception of an application than almost any other component in the stack. Unfortunately, it's often impractical to monitor performance directly *on* the client, particularly in a commercial application setting where the client belongs to your customer and not to you.

You can, of course, have a test client computer with hardware and software configurations that resemble your average client computer or customer computer, and you can install monitoring software on that computer to see what kind of performance problems, if any, the client is introducing into the equation. That doesn't always help you troubleshoot problems at the client level. In fact, in some cases, client-specific problems *can't* practically be solved. For example, suppose some of your customers run a particular brand of antivirus software that simply slows their computers—there's nothing you can really do about that, aside from making sure that your client application performs as best it can under the circumstances.

There's unfortunately not much else to say about the client layer. It's often outside your control, and although you can and should test your client applications on representative client computers, that's about all you can do in a traditional setting.

Network Layer

At the network layer, however, you can begin to get more involved. You can *certainly* monitor your network's performance, and many tools exist to help you do so. For example, Figure 4.2 shows a tool that helps to monitor network performance at a particular network node.

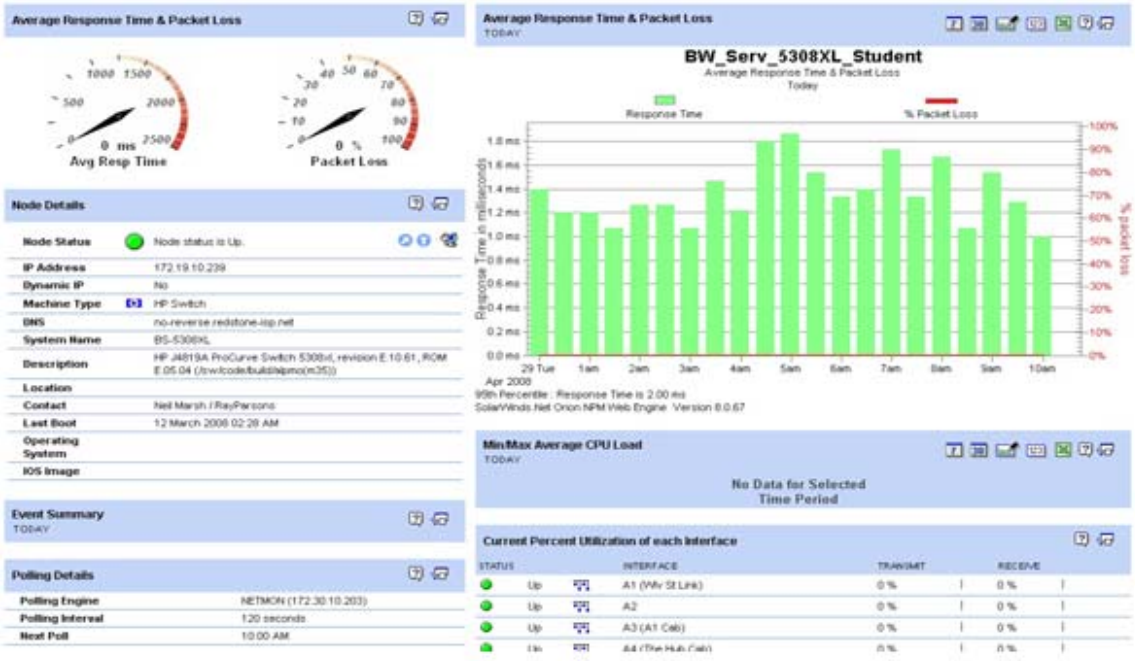


Figure 4.2: Monitoring network performance.

For a larger view of your entire network, there are tools to roll up per-node performance monitoring into a "whole network" view, as illustrated in Figure 4.3. But there's a problem with this kind of monitoring: It's absolutely unaware of the *application*. This type of infrastructure-level monitoring can tell you whether a particular router is overloaded, for example, but it can't correlate that problem to a particular application performance issue.

Figure 4.3: Monitoring the entire network.

Another problem with these types of tools is that they start to fail you once you begin relying on *someone else's* network. As you move toward a hybrid IT environment, and you begin deploying elements of applications in the cloud—meaning in someone else's data centers—you lose the ability to closely track network device performance.

I would argue that, although this type of monitoring tool is often widely-used within organizations, it isn't really all that useful for application performance monitoring because it isn't application aware—it's simply reporting the state of the network. Also, as applications begin to move toward a hybrid or cloud-based model, this type of tool simply loses all its utility, as it can't do its job across a super-distributed, partially-outsourced network.

Application and Database Layers

As you move into the application itself, performance monitoring can become even *more* complicated. Ignoring the hardware that connects and runs the application, modern applications consist of numerous interconnected components—many of which may be shared by other applications. This is especially true in a cloud computing scenario: If you're storing data in Microsoft's SQL Server for Windows Azure, there are likely multiple other customers doing the same thing. Thus, your performance could potentially be impacted by others' use of the cloud infrastructure. That kind of sharing can even occur within applications that live entirely within your own data center, making performance troubleshooting complex and difficult. Consider a relatively straightforward Web application, as Figure 4.4 shows.

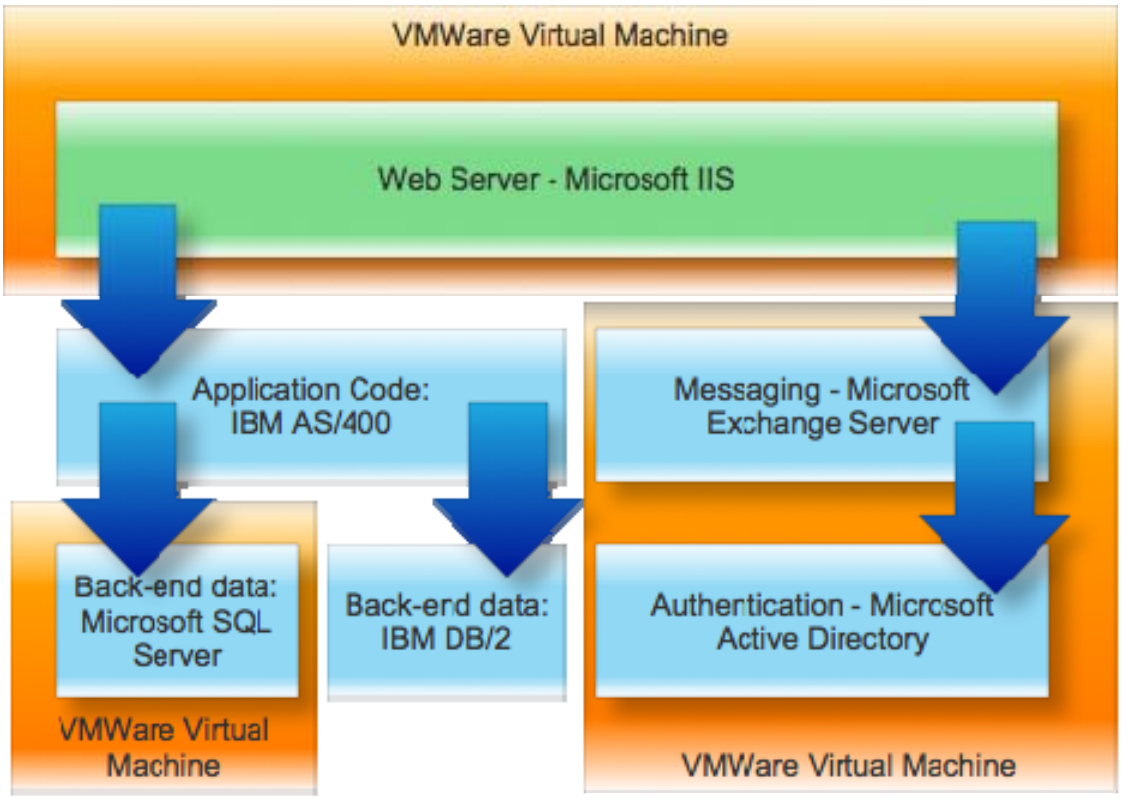


Figure 4.4: Application software stack.

This application consists of a Web server, application code running on an AS/400, and back-end data from two different databases. The application itself utilizes Exchange Server for messaging, which in turn has a dependence on Active Directory (AD) for authentication, address books, and so on. Much of the application is running inside VMware virtual machines, which add another layer of performance complexity. How do you monitor an application like this?

Traditionally, you'd probably need about seven tools, each one to monitor a specific component: IIS, VMware, AD, Exchange, DB/2, the AS/400, and SQL Server. Each tool would be entirely focused on a single element of the application stack. For example, Figure 4.5 shows what you might expect from a VMware monitoring solution: information on CPU, disk, memory, and network utilization.

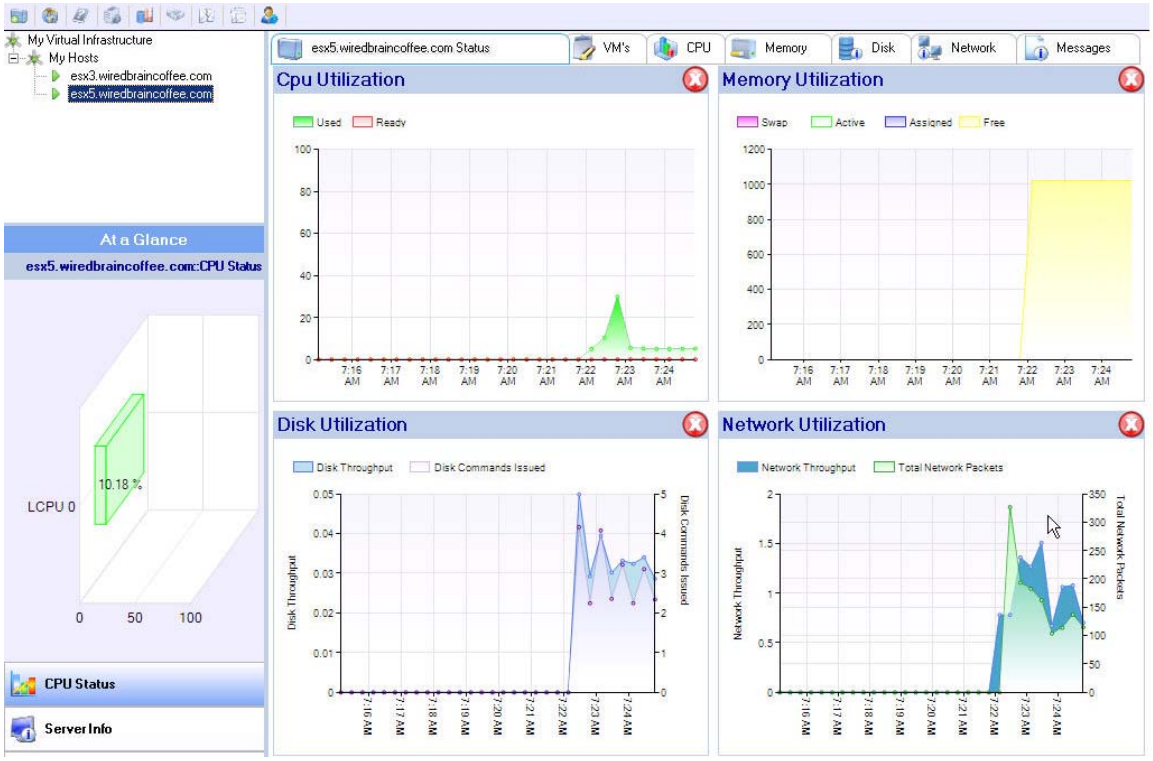


Figure 4.5: VMware monitoring.

A SQL Server monitoring tool, in contrast, would present a completely different set of statistics and views—and wouldn't be at all aware of the underlying impact from VMware itself. As Figure 4.6 shows, you'd be dealing with two completely different tools, with different goals and methodologies—neither of which would have the slightest idea about your *application's* performance.

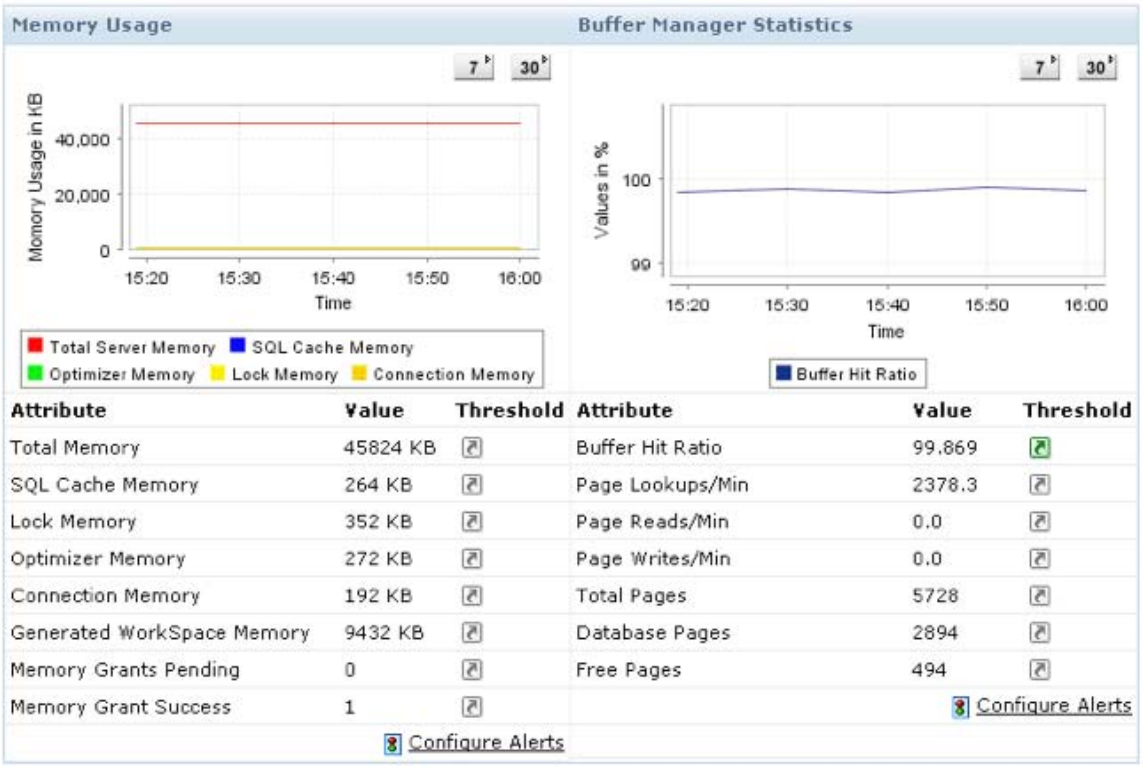


Figure 4.6: Monitoring SQL Server performance.

That's ultimately the problem with traditional application performance monitoring even *before* you start involving hybrid IT elements like cloud computing: every tool is focused on one bit of the application, and those tools don't even monitor *the application.* Those tools just monitor their one element, in a completely standalone and out-of-context fashion. Start involving cloud computing and things get even more difficult because you aren't likely to even get a tool that will let you directly monitor your cloud database performance.

Other Concerns

So let me clearly state the problem: **Performance tools that focus only on a single element of the application are ineffective.** In fact, that can actually *hinder* your application performance monitoring and troubleshooting efforts, as you'll see in the next section. And that's assuming your applications are entirely under your control, within your own data center; start moving application elements into the cloud—whether as Software as a Service (SaaS) solutions, hosted services, or true cloud computing—and these kinds of tools become even more impractical because in many cases you can't use them to monitor the cloud elements of your application.

Multi-Discipline Monitoring and Troubleshooting

Part of the problem with element-specific tools like those I discussed earlier is that they encourage "siloing" within your IT organization. IT professionals tend to specialize in just one or two elements of an application, and the fact that their tools only monitor those elements enables them to put on blinders for the rest of the application. That simply causes more difficulty when performance problems arise.

Applications Are Not the Sum of Their Parts

The problem is that you can never look at a single element of an application and judge its overall performance. That's a very intuitive concept that you might not have actively thought about, but do so for a second: Do you think you could look strictly at a single application element—say, the database, the Web server, or a network router—and determine the overall performance of the application? No, of course not. Therefore, you can't look at *every* component in a standalone fashion and determine performance information, either. You can't take a bunch of disparate statistics from multiple tools and merge them in your head for a "performance picture" of an application—it just isn't practical.

That's because applications aren't simply the sum of their parts. In other words, you can't simply add up or average the performance numbers for various application elements and arrive at an accurate top-level performance number for the entire application.

Application performance is, in this respect, a lot like a sports team. You can't just add up the average performance of each player and get a feel for the team's overall success. Nor can you simply look for the worst-performing player and focus all your troubleshooting efforts on that person. Individuals might perform very well on their own in practice, and perform entirely differently when the whole team is in action. You have to manage the team's performance *as a team* by watching the *entire* group of players work together. When it comes to an individual's performance, you might coach them to adopt certain behaviors or to correct specific problems. Doing so, however, might not change their *interactions* with other team members during actual play. You can't simply "tune" an individual player on their own; you need to "tune" everyone's performance as a part of the team.

Tossing Problems Over the Fence: Troubleshooting Challenges

Tuning individual application elements is what IT professionals tend to do well, but it can result in significant delays and dead-ends when it comes to managing the performance of an *entire application.* The week before I wrote this chapter, for example, I visited a consulting client who was experiencing performance problems with an application. I sat down with representatives from their various IT disciplines, and had a conversation something like this:

> **Me:** So what's the problem?
>
> **Manager:** We're seeing a slowdown in one of our applications. We've traced the problem to a particular query for data from the database—sometimes, it can take more than 4 minutes to execute that one query.
>
> **DBA:** I've looked at the SQL Server performance, though, and there doesn't seem to be a problem. Anytime I run that same query manually, it works fine. I've rebuilt the indexes on the affected tables. I've even run the query repeatedly, using a load-simulation tool, in a test environment and it works fine.
>
> **Network Engineer:** We're not seeing anything in the network. Whenever someone reports the slowdown in the application, the network is at the exact same performance levels it always is.
>
> **Developer:** It is definitely not the client application. We've tested it extensively. Anytime this delay occurs, it happens as the client application is waiting for data to return from the query. The application pauses, but it isn't doing anything—it's just waiting on the database.
>
> **DBA:** It can't be waiting on the database; I'm not seeing any indication that the database is taking more than a few milliseconds to process that query! It must be the amount of data you're querying.
>
> **Developer:** No, it isn't—we're only grabbing two rows of data, three at most.

That continues for a half-hour or so, with all sides offering up charts and other evidence that their element wasn't causing the problem. So if nothing was causing the problem, what was the problem? This is a classic example of what I call "tossing the problem over the fence." Each individual IT discipline has set itself up in a fenced-off silo, and they only concern themselves with what's happening inside their fence. If they determine, in their own judgment, that their bit of the world is fine, then they toss the problem over the fence to another discipline.

That conversation was about an application living entirely within the company's data center: Just imagine how much more fun it would have been if outside vendors—SaaS providers, Managed Service Providers (MSPs), cloud computing vendors, and so on—were involved! With absolutely no insight into the outsourced components' performance, everyone within the organization would likely have tossed the problem right outside the organization and into the laps of those vendors. Unless those vendors had tools that could show *they* weren't the problem, the argument would have gone on forever.

So what's the solution? *Integrated* monitoring, or what some would call *unified* monitoring. You still have to monitor the individual application elements; you just do so *all in one place.*

Integrated, Bottom-Up Monitoring

The idea behind integrated, or unified, monitoring is to give everyone access to the same information, in the same way, in the same place. You're still monitoring components, to be sure—each component does contribute to or take away from the overall application performance, after all. But you're doing so in a way that puts all the information in front of all the experts at the same time. That helps to break down the fences between IT disciplines, gives everyone the same evidence to work from, and helps to integrate the performance troubleshooting effort more effectively.

Monitoring Performance Across the Entire Stack

Consider a completely unified dashboard, like the one shown in Figure 4.7. Here, you can get a top-level view for *all* your application components—not just one, and without the need to open multiple tools. You can establish thresholds for "good" and "bad" performance, and get a quick at-a-glance view of how your components are all doing. Anything other than green in any one spot may indicate a performance problem that will impact your EUE.

Figure 4.7: Dashboard of all components.

When you do see something other than green, you need the ability to drill-down into component-specific performance information. For example, the VMware and Hyper-V status for this application is orange, meaning there's some kind of problem. Drilling-down might reveal a screen like the one in Figure 4.8, which breaks down the problem in domain-specific terms. The dashboard lets everyone see *where* the problem lies; the drill-down helps to start the troubleshooting process. There's no need to toss anything over any fences, because it's clear where the problem exists.

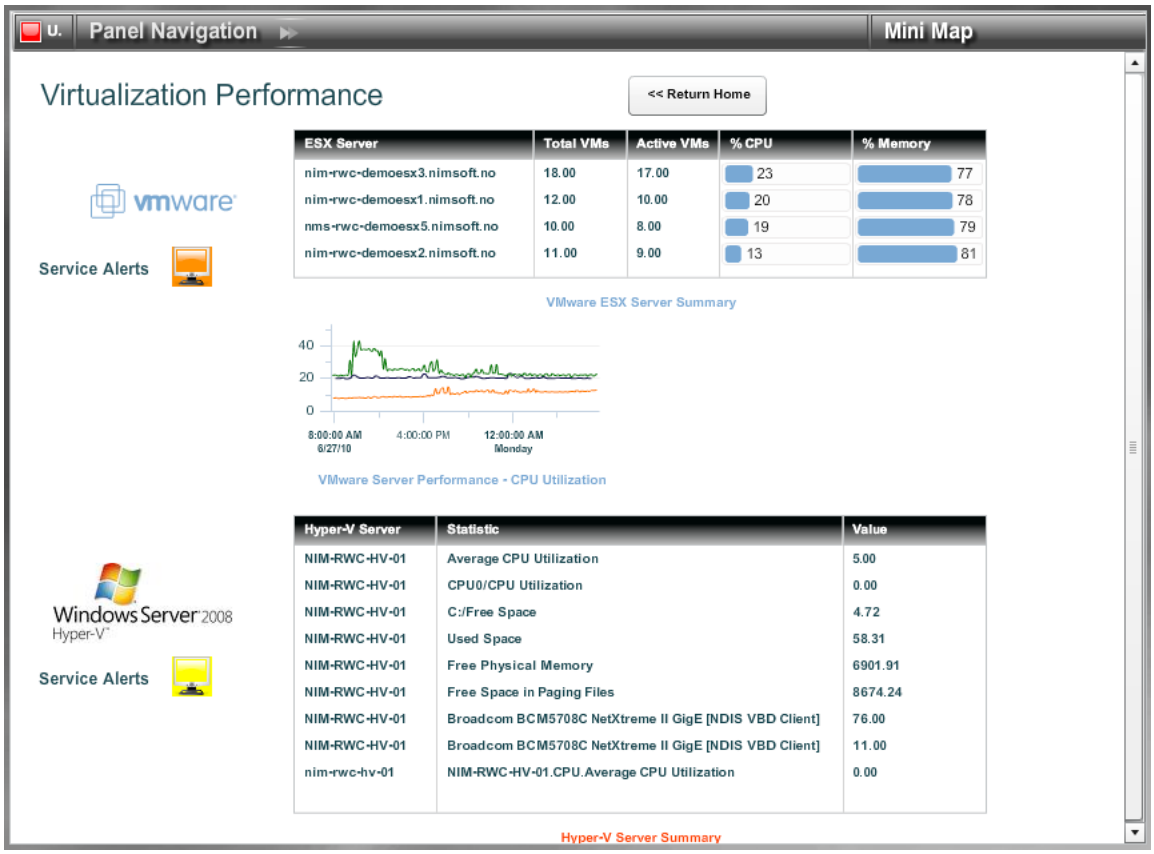


Figure 4.8: Detail for virtualization problem.

Here, we can see that there specific service alerts for both VMware and Hyper-V. By viewing those alerts, we can begin to troubleshoot the problem more quickly. We've gotten the right domain expert involved, made it clear that there's no over-the-fence option because we *know* the problem is in his or her domain, and have gotten troubleshooting started.

The key to a toolset like this is having *every* part of the application represented. As Figure 4.9 shows, that representation can even include infrastructure components—in this case, Cisco-powered Voice over IP (VoIP) services.

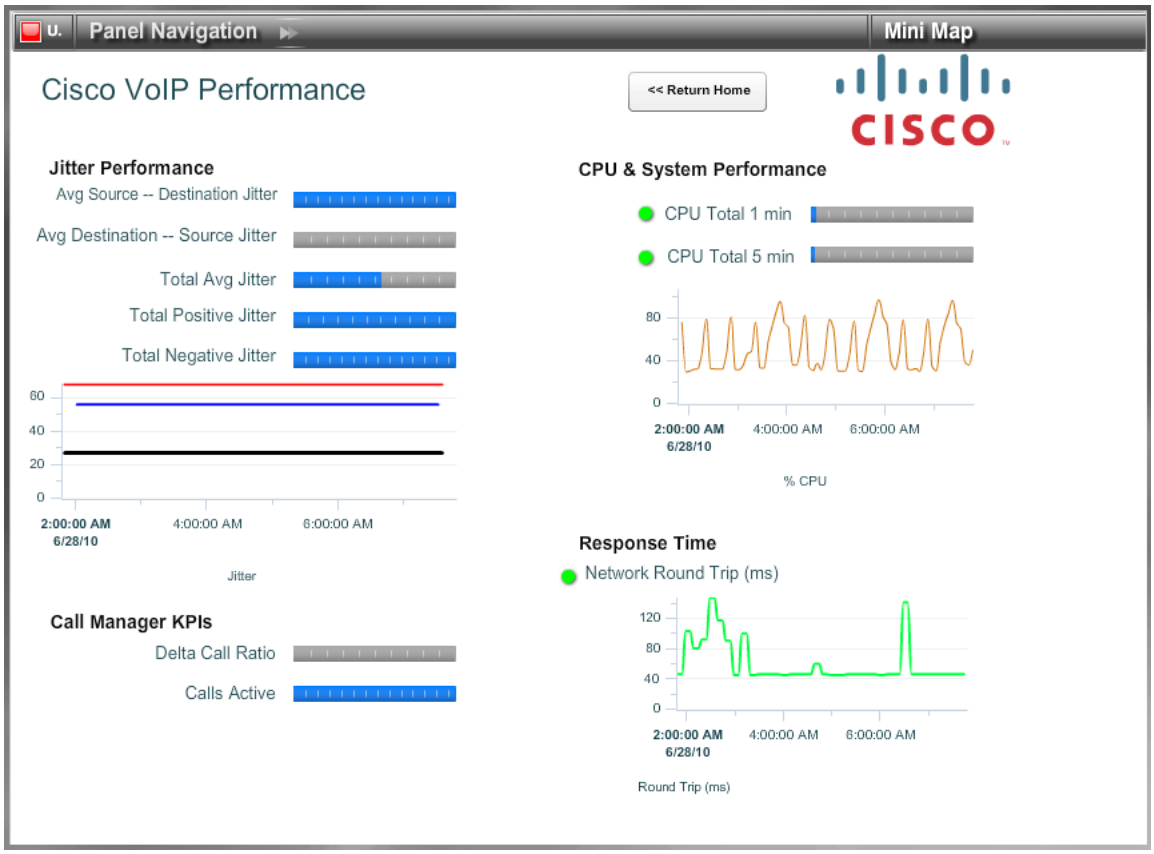


Figure 4.9: Including infrastructure services in the monitoring.

It's critical that we be able to see performance for *everything that the application depends upon.* That way, no IT discipline is excluded or forgotten, and there aren't any "hidden dependencies" impacting performance under the hood or behind the scenes. As you can see from Figure 4.9, however, including everyone doesn't mean making the performance information generic: This example clearly illustrates the domain-specific details that a tool must be able to deliver. VoIP is entirely different from, say, a database server, and the troubleshooting tools have to respect that difference and represent appropriate information for each element.

Our own data center needs to be included (see Figure 4.10). The idea is to roll up all the resources under *our* control so that we can get a top-level performance or health metric for our self-managed resources. This drill-down lets us see network devices, individual servers, virtualization hosts, and the other elements under our direct control. When something exceeds a threshold, we can immediately dispatch the right domain expert to begin troubleshooting the problem—and we would have some confidence that the problem *is* on our end and within our control.

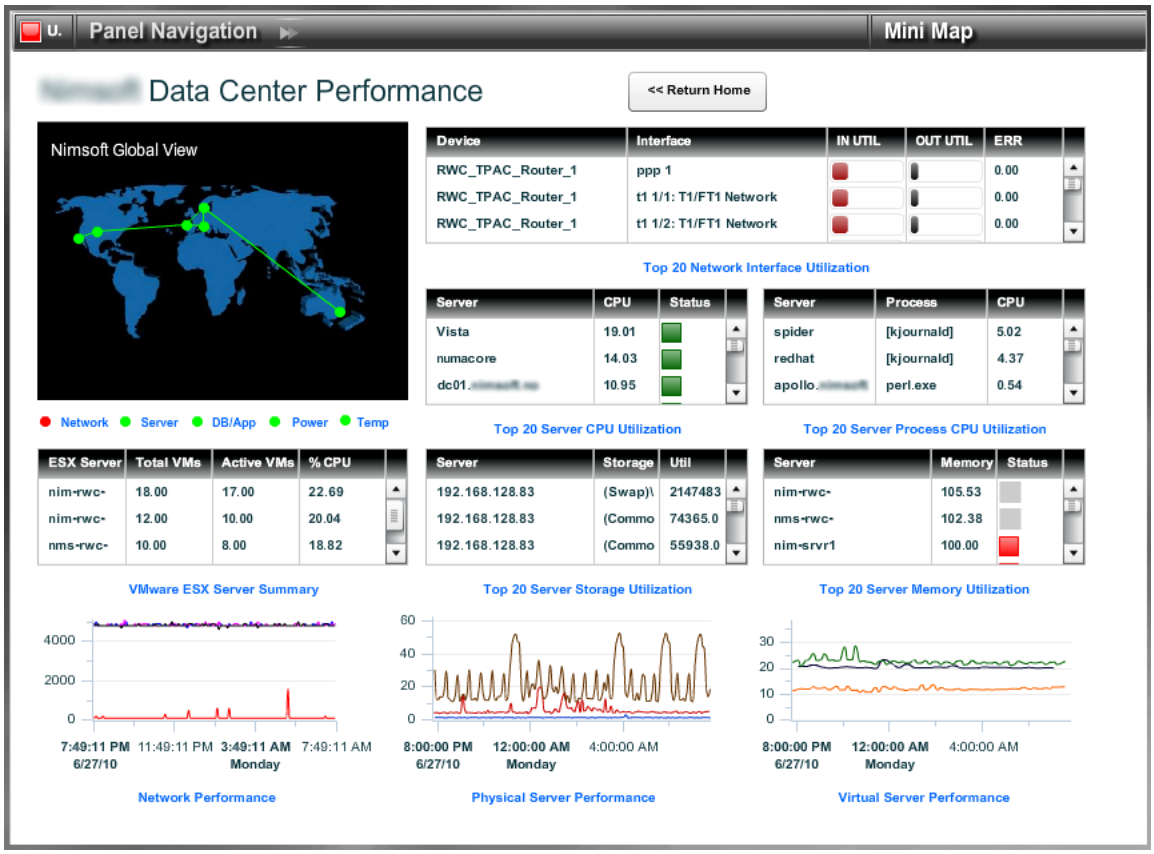


Figure 4.10: Including our data center in monitoring.

The corollary, of course, is that the monitoring solution also needs a way to monitor the resources *not* under our direct control. As Figure 4.11 shows, that might include cloud computing services like Amazon Web Services. This is where a monitoring solution really needs to break with traditional techniques: Instead of monitoring these external services *directly,* the tool must often do so through vendor-supported application programming interfaces (APIs), through direct observation of service response times, and so forth. This is really a new field in performance monitoring, so as you begin evaluating solutions, it will be important to consider different vendors' ability to draw information from the outsourced services that you rely upon to run your application.

Figure 4.11: Monitoring cloud computing services.

I have to emphasize *tight* integration with outsourced services. For example, in the original dashboard view, you'll notice that SalesForce.com shows a status color of blue—not the green we'd hope for, but not as immediately alarming as yellow, orange, or red. Drilling-down reveals a service alert, shown in Figure 4.12. As you begin working with services outside your own data center, you need to become more aware of *their* data center operations—including, in this case, a scheduled maintenance window that may result in diminished performance for our application that relies on SalesForce.com. By having this information *right within the monitoring console,* we can help manage our *future* performance more effectively. We might choose to temporarily turn off portions of our application that depend on SalesForce.com during that maintenance window, or we might simply need to be alert for performance problems that result from the maintenance window.

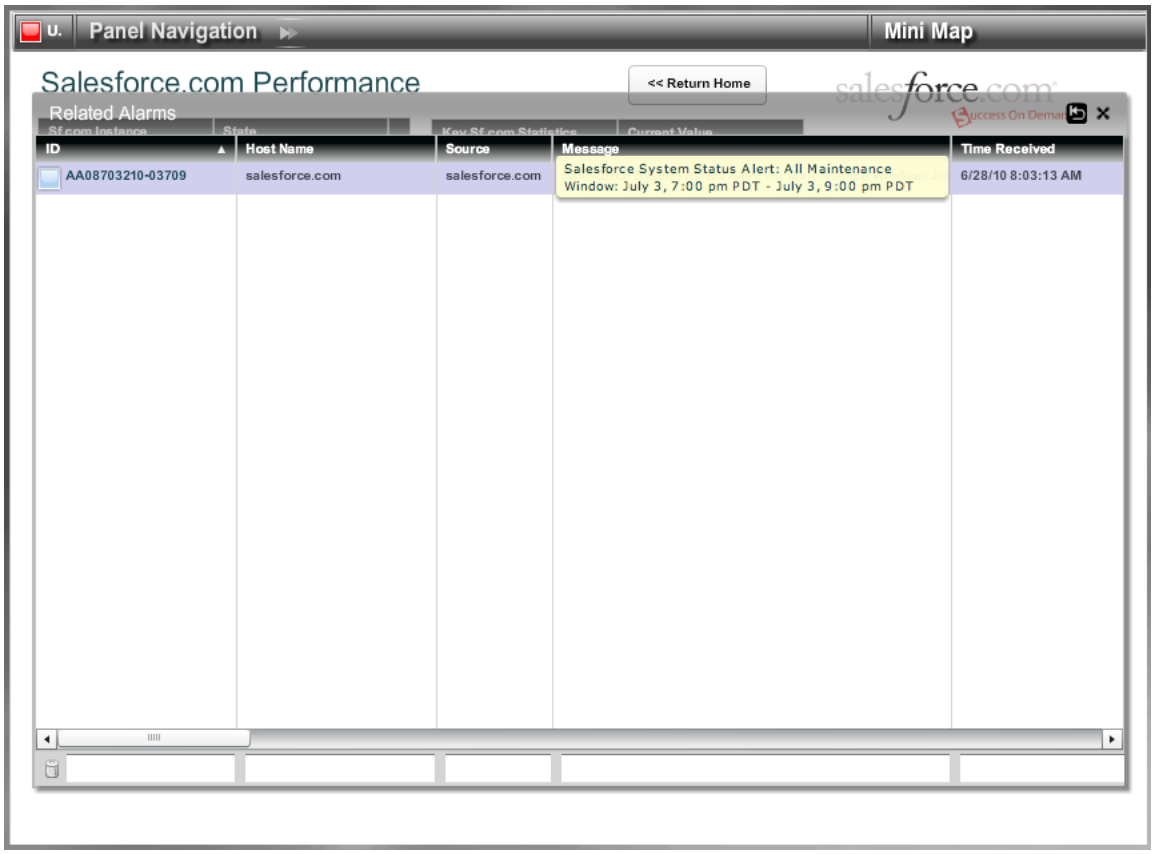


Figure 4.12: Viewing service alerts.

As you move toward a hybrid IT model, this kind of integration with external service providers will prove absolutely invaluable.

Integrated Troubleshooting Saves Time and Effort

I presented this unified monitoring concept to the client I was with, and they gave it a shot. As I was writing this chapter, they contacted me to let me know they had deployed a unified monitoring solution and that they were quite happy with it. It turns out the problem *was* in the SQL Server, and had something to do with the way the server was recompiling that query. The DBA was, unfortunately, a bit stubborn about admitting it, but with a single tool showing perfect performance in all but his application component, he was forced to start troubleshooting the problem. By getting the same tool in front of everyone, each discipline's fences started to come down a bit. Sure, they were all responsible for their individual elements, but it gets a lot harder to toss a problem over the fence when *everyone* can clearly see that it's on your side.

The Provider Perspective: Providing Details on Your Stack

MSPs have a more challenging time of monitoring. They must not only monitor their own data centers—contending with all the multi-discipline problems I've described in this chapter—but also provide their customers with a rolled-up view of their services. Ideally, they should do so in a way that customers can integrate into *their* monitoring tools so that service alerts and other information "rolls down" from the MSP's data center into the customers' monitoring tools.

With the right monitoring tools, you can do that pretty easily. What you're really after is a tool that gives you all the detailed, cross-discipline, unified monitoring you want *within* your data center, with the ability to aggregate some of that data into a "status indicator" *for* your data center, done in such a way that your aggregate indicator can become a part of your *customers'* dashboards. Figure 4.13 illustrates the concept.

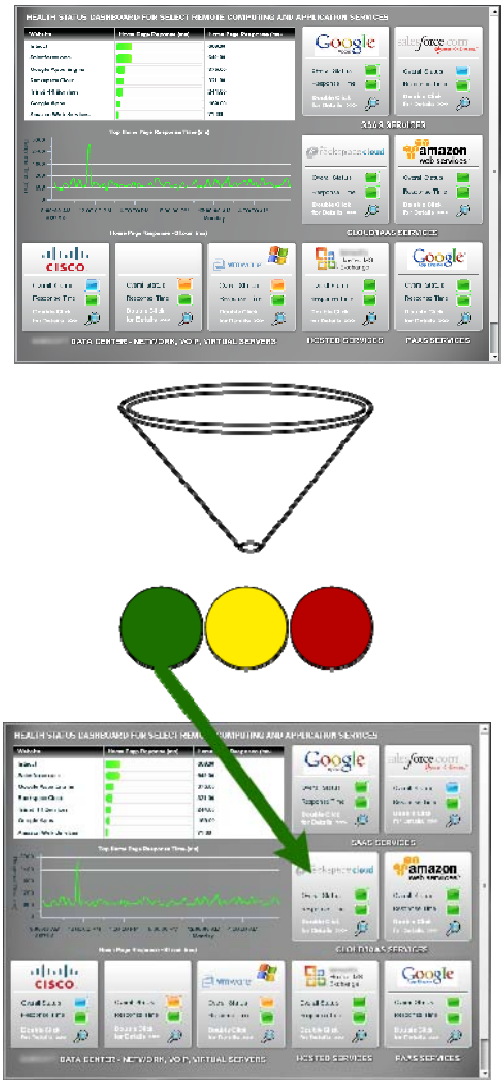


Figure 4.13: Aggregating your MSP network for customer information.

Of course, you're quite likely going to want to provide your customers with more detailed information as well—because they're probably going to demand it: custom dashboards specific to your service offerings, quality of service (QoS) reports, SLA reports, and more. The last chapter of this book will dive more into those offerings.

Coming Up Next…

You should now have a vision for how your cloud-based or hybrid applications can be monitored from the EUE and the component level. In the next chapter, we'll start exploring some of the specific capabilities you need to start monitoring a hybrid IT environment—from your data center into the cloud. Consider the next chapter to be a sort of "shopping list" layout of all the features that you should at least be considering in a monitoring solution.

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.