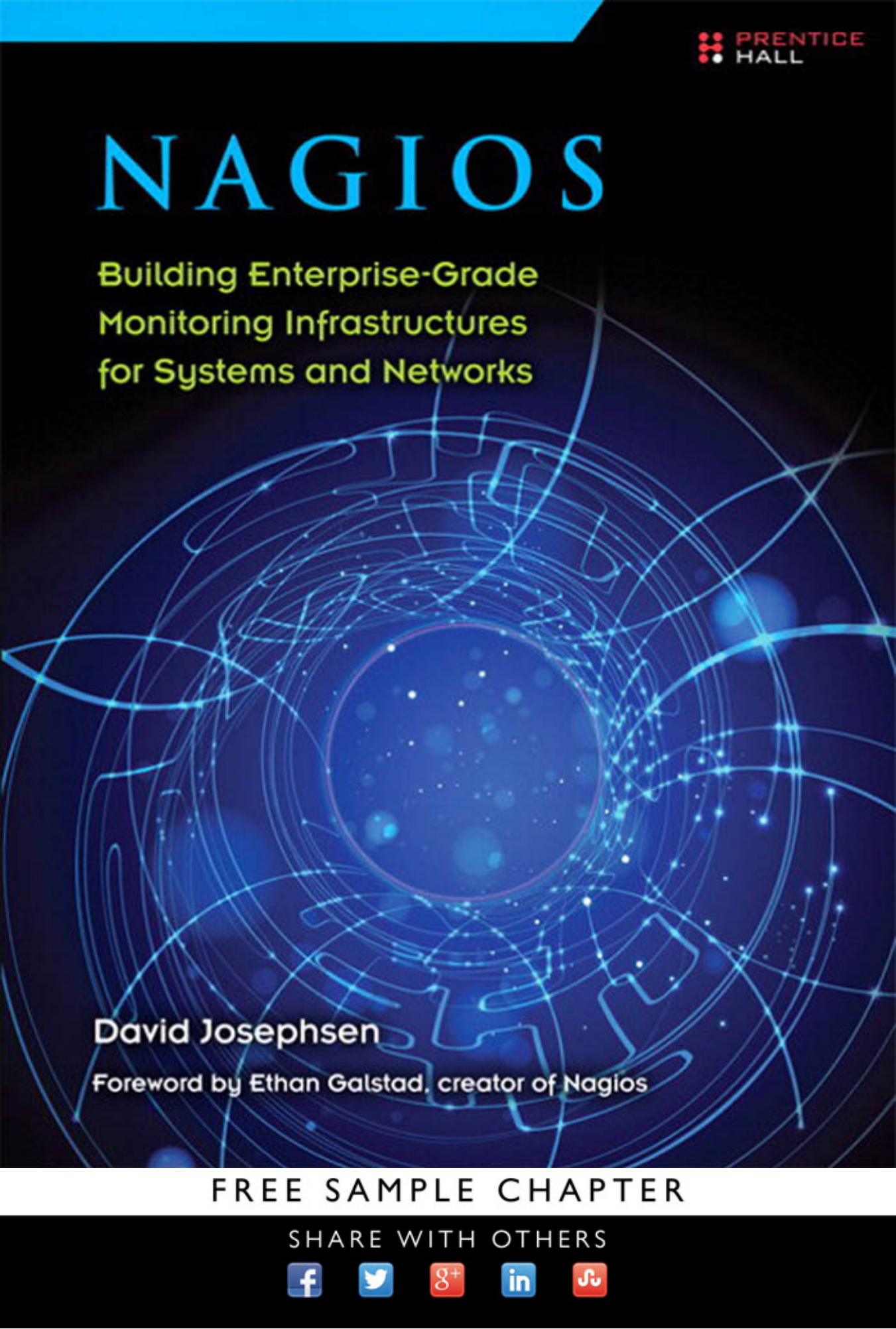
PRENTICE HALL
NAGIOS
Building Enterprise-Grade Monitoring Infrastructures for Systems and Networks
David Josephsen
Foreword by Ethan Galstad, creator of Nagios
FREE SAMPLE CHAPTER
SHARE WITH OTHERS

Nagios

This page intentionally left blank

Nagios

Building Enterprise-Grade Monitoring Infrastructure for Systems and Networks

Second Edition

David Josephsen

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris
Madrid • Cape Town • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data is on file.

ISBN-13: 978-0-13-313573-2
ISBN-10: 0-13-313573-X

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.

First Printing: April 2013

For Cynthia, for enduring and encouraging my incessant curiosity.
And for Tito, the cat with the biggest heart.

This page intentionally left blank

CONTENTS

This page intentionally left blank

FOREWORD

People often say that Nagios is "flexible," by which I think they mean that it is easily extended, but that misses the point. The power inherent in Nagios' design derives not from its extensibility, but rather from its insistence on being extended. This is an admittedly small but important distinction. Many pieces of software can be extended to do new things, but very few pieces of software do nothing until you've extended them, and it is exactly because of this—this inherent demand that you customize it to suit your needs—that Nagios has always been a synthesis of contributions from engineers and administrators working to solve their own individual problems. No two installations are alike, and that is by design.

In the years since I first created Nagios, it has grown in breadth and scope beyond anything I'd imagined. With over 1 million users worldwide, Nagios Core has found a home everywhere from huge Fortune-500 conglomerates to state of the art scientific research labs. The Nagios user community is one of the healthiest and most actively contributing open source communities out there, with nearly 4,000 published plug-ins, add-ons, and extensions—many of which are sufficiently complex to warrant books of their own. The community is so large, diverse, and active, that Nagios now has its own annual conference where contributors, users, and educators come together to share ideas, learn tips and tricks, and find out about upcoming developments in the project.

There is also a thriving community of corporations at work on extending and supporting Nagios. In 2007 I joined them, founding Nagios Enterprises. Our flagship product, Nagios XI, is both an evolutionary step forward, and (as it should be) a fully-reverse compatible extension to Nagios Core. XI embraces the extend-by-design lineage of Core, preserving the power and flexibility of Core, while expanding its accessibility and usability.

But even given the wonderful success Nagios has enjoyed, I'm the first to admit that flexibility comes with a price. It can be difficult for newcomers and experienced admins alike to build and deploy a successful monitoring solution, and many of the challenges have nothing whatsoever to do with computers. Luckily, David is one of the few technical writers that are able to cover a complex subject like this in an easy-to-understand format. Whether you're a newcomer to the world of network, system, and IT monitoring, or you're an experienced Nagios admin, David's work is sure to be helpful to you.

—Ethan Galstad, Nagios Founder and President

This page intentionally left blank

ACKNOWLEDGMENTS

My lovely wife, Cynthia, is patient and encouraging and pretty, and I love her.

Ethan Galstad, whose interest prompted the second edition, and without whom there would be no Nagios.

The tech reviewers on this project were outstanding—thanks, guys.

Last, my editors at Prentice Hall have been great. They aren't at all like the editors in Spiderman or Fletch. Debra Williams Cauley and Kim Boedigheimer are a hardworking, on the ball, and clued-in pair of professionals. They've been patient and helpful, and I appreciate their time and attention.

Thanks.

This page intentionally left blank

ABOUT THE AUTHOR

David Josephsen is the Director of Systems Engineering at DBG, Inc., where he maintains a collection of geographically dispersed server farms. He has more than a decade of hands-on experience with UNIX systems, routers, firewalls, and load balancers in support of complex, high-volume networks. In addition to this book, he authored several chapters in the O'Reilly book *Monitoring with Ganglia*, and currently writes "iVoyer," the systems monitoring column for *;login* magazine. Josephsen is just one of many thousands of avid Nagios users.

This page intentionally left blank

ABOUT THE TECHNICAL REVIEWERS

Mark Bainter

Mark Bainter leads a team of sysadmins providing outsourced monitoring and management of high volume mail systems for Message Systems' clients, leveraging over 15 years experience as a sysadmin specializing in systems integration, monitoring, and automation. He is an autodidactic polymath and impenitent sesquipedalian. Mark currently resides in Texas with his lovely wife and four children and in his free time he enjoys reading, woodworking, and losing at Settlers to his wife.

Mike Guthrie

Mike Guthrie is the lead developer at Nagios enterprises and has developed new features and add-ons for Nagios Core, Nagios XI, and Nagios Fusion. Mike does the bulk of his programming in PHP and particularly enjoys front-end web development and data visualizations. When he's not at work, he enjoys spending time with his family, being outside, and working on his house.

Mathias Kettner

Mathias Kettner is known as the author of Check_MK, MK Livestatus, and other Nagios add-ons. He runs a fast growing company in Munich, Germany, which is dedicated to system monitoring based on Nagios, and offers professional support and software development.

This page intentionally left blank

Introduction

This is a book about untrustworthy machines—machines, in fact, that are every bit as untrustworthy as they are critical to our well being. But I don't need to bore you with a laundry list of how prevalent computer systems have become or with horror stories about what can happen when they fail. If you picked up this book, I'm sure you're well aware of the problems: layer upon layer of interdependent libraries hiding bugs in their abstraction, script kiddies, viruses, DDOS attacks, hardware failure, end-user error, backhoes, hurricanes, and on and on. It doesn't matter whether the root cause is malicious or accidental; your systems will fail, and when they do, only two things will save you from the downtime: redundancy and monitoring systems.

Do It Right the First Time

In concept, monitoring systems are simple: an extra system or collection of systems whose job is to watch the other systems for problems. For example, the monitoring system could periodically connect to a Web server to make sure it responds and, if not, send notifications to the administrators. Although it sounds straightforward, monitoring systems have grown into expensive, complex pieces of software. Many now have agents larger than 500MB, include proprietary scripting languages, and sport price tags above $60,000.

When implemented correctly, a monitoring system can be your best friend. It can notify administrators of glitches before they become crises, help architects tease out patterns corresponding to chronic interoperability issues, and give engineers detailed capacity planning information. A good monitoring system will help the security guys correlate interesting events, show the network operations center personnel where the bandwidth bottlenecks are, and provide management with much needed high-level visibility into the critical systems that they bet their business on. A good monitoring system can help you uphold your service level

agreement (SLA) and even take steps to solve problems without waking anyone up at all. Good monitoring systems save money, bring stability to complex environments, and make everyone happy.

When done poorly, however, the same system can wreak havoc. Bad monitoring systems cry wolf at all hours of the night so often that nobody pays attention anymore; they install backdoors into your otherwise secure infrastructure, leech time and resources away from other projects, and congest network links with megabyte upon megabyte of health checks. Bad monitoring systems can really suck.

Unfortunately, getting it right the first time isn't as easy as you might think, and in my experience, a bad monitoring system doesn't usually survive long enough to be fixed. Bad monitoring systems are too much of a burden on everyone involved, including the systems being monitored. In this context, it's easy to see why large corporations and governments employ full-time monitoring specialists and purchase software with six-figure price tags. They know how important it is to get it right the first time.

Small- to medium-sized businesses and universities can have environments as complex as or even more complex than large companies, but they obviously don't have the luxury of high-priced tools and specialized expertise. Getting a well-built monitoring infrastructure in these environments, with their geographically dispersed campuses and satellite offices, can be a challenge. But having spent a good part of the past 13 years building and maintaining monitoring systems, I'm here to tell you that not only is it possible to get it done right the first time, but you can do it for free, with a bit of elbow grease, some open source tools, and a pinch of imagination.

Why Nagios?

Nagios is, in my opinion, the best system and network monitoring tool available, open source or otherwise. Its modularity and straightforward approach to monitoring make it easy to work with and highly scalable. Further, Nagios's open source license makes it freely available and easy to extend to meet your specific needs. Instead of trying to do everything for you, Nagios excels at interoperability with other open source tools, which makes it very flexible. If you're looking for a monolithic piece of software with check boxes that solve all your problems, this probably isn't the book for you. But before you stop reading, give me another paragraph or two to convince you that the check boxes aren't really what you're looking for.

Most commercial offerings get it wrong because their approach to the problem assumes that everyone wants the same solution. To a certain extent, this is true. Everyone has a large glob of computers and network equipment and wants to be notified if some subset of it fails.

So if you want to sell monitoring software, the obvious way to go about it is to create a piece of software that knows how to monitor every conceivable piece of computer software and networking gear in existence. The more gadgets your system can monitor, the more people you can sell it to. To someone who wants to sell monitoring software, it's easy to believe that monitoring systems are turnkey solutions and whoever's software can monitor the largest number of gadgets wins.

The large commercial packages I've worked with all seem to follow this logic. Not unlike the Borg, they are methodically locating new computer gizmos and adding the requisite monitoring code to their solution—or worse, acquiring other companies who already know how to monitor lots of computer gadgetry and bolting those companies' code onto their own. They quickly become obsessed with features, creating enormous spreadsheets of supported gizmos. Their software engineers exist so that the presales engineers can come to your office and say to your managers, through seemingly layers of white gleaming teeth, "Yes, our software can monitor that."

The problem is that monitoring systems are not turnkey solutions. They require a large amount of customization before they start solving problems and herein lies the difference between people selling monitoring software and those designing and implementing monitoring systems. When you're trying to build a monitoring system, a piece of software that can monitor every gadget in the world by clicking a check box is not as useful to you as one that makes it easy to monitor what you need, in exactly the manner that you want. By focusing on *what* to monitor, the proprietary solutions neglect the *how,* which limits the context in which they may be used.

Take ping, for example. Every monitoring system I've ever dealt with uses ICMP Echo requests, otherwise known as pings, to check host availability in one way or another. But if you want to control *how* a proprietary monitoring system uses ping, architectural limitations become quickly apparent. Let's say I want to specify the number of ICMP packets to send, or I want to be able to send notifications based on the round-trip time of the packet in microseconds instead of simple pass/fail. More complex environments may necessitate that I use IPv6 pings, or that I portknock[1] before I ping. The problem with the monolithic, feature-full approach is that these changes represent changes to the core application logic and are, therefore, nontrivial to implement.

In the commercial monitoring applications I've worked with, if these ping examples could be performed at all, they would require reimplementing the ping logic in the monitoring system's proprietary scripting language. In other words, you would have to toss out the built-in ping functionality altogether. Perhaps being able to control the specifics of ping checks is of questionable value to you, but if you don't have any control over something as basic as ping, what are the odds that you'll have finite enough control over the most important checks

in your environment? They've made the assumption that they know *how* you want to ping things and from then on it was game over; they never thought about it again. And why would they? The ping feature is already in the spreadsheet, after all.

When it comes to gizmos, Nagios's focus is on modularity. Single-purpose monitoring applets called plug-ins provide support for specific devices and services. Rather than participating in the feature arms race, hardware support is community driven. As community members have a need to monitor new devices or services, new plug-ins are written and usually more quickly than the commercial applications can add the same support. In practice, Nagios will always support everything you need it to and without ever needing to upgrade Nagios itself. Nagios also provides the best of both worlds when it comes to support, with several commercial options, as well as a thriving and helpful community that provides free support through various forums and mailing lists.

Choosing Nagios as your monitoring platform means that your monitoring effort will be limited by your own imagination, technical prowess, and political savvy. Nagios can go anywhere you want it to and the trip there is usually pretty simple. Although Nagios can do everything the commercial applications can, and more, without the bulky, insecure agent install, it usually doesn't compare favorably to commercial monitoring systems because when spreadsheets are parsed, Nagios doesn't have as many checks. In fact, if they're counting correctly, Nagios has no checks at all, because technically it doesn't know *how* to monitor anything; it prefers that you tell it how. The question of "how" is difficult to encompass with a check box.

What's in This Book?

Although Nagios is the biggest piece of the puzzle, it's only one of the myriad of tools that make up a world-class open source monitoring system. With several books, superb online documentation, and lively and informative mailing lists, it's also the best-documented piece of the puzzle. So my intention in writing this book is to pick up where the documentation leaves off. This is not a book about Nagios as much as it is a book about the construction of monitoring systems using Nagios, and there is much more to building monitoring systems than configuring a monitoring tool.

I'll cover the usual configuration boilerplate, but configuring and installing Nagios is not my primary focus. Instead, to help you build great monitoring systems, I need to introduce you to the protocols and tools that enhance Nagios's functionality and simplify its configuration. I need to give you an in-depth understanding of the inner workings of Nagios itself, so you can extend it to do whatever you might need. I need to spend some time in this book exploring possibilities because Nagios is limited only by what you feel it can do.

Finally, I need to write about things only loosely related to Nagios, like best practices, SNMP, visualizing time-series data, and various Microsoft scripting technologies, such as WMI and WSH.

Most important, I need to document Nagios itself in a different way. By introducing it in terms of a task-efficient scheduling and notification engine, I can keep things simple while talking about the internals up front. Rather than relegating important information to the seldom-read advanced section, I'll empower you early by covering topics like plug-in customization and scheduling as core concepts.

Although the chapters more or less stand on their own, and I've tried to make the book as reference-friendly as possible, I think it reads better as a progression from start to finish. I encourage you to read from cover to cover, skipping over anything you are already familiar with. The text is not large, but I think you'll find it dense with information and even the most seasoned monitoring veterans should find more than a few useful nuggets of wisdom.

The chapters tend to build on each other and casually introduce Nagios-specific details in the context of more general monitoring concepts. Because many important decisions need to be made before any software is installed, I begin with "Best Practices" in Chapter 1. This should get you thinking in terms of what needs to take place for your monitoring initiative to be successful, such as how to go about implementing, who to involve, and what pitfalls to avoid.

Chapter 2, "Theory of Operations," builds on Chapter 1's general design guidance by providing a theoretical overview of Nagios from the ground up. Rather than inundating you with configuration minutiae, Chapter 2 will give you a detailed understanding of how Nagios works without being overly specific about configuration directives. This knowledge will go a long way toward making configuration more transparent later.

Before we can configure Nagios to monitor our environment, we need to install it. Chapter 3, "Installing Nagios," should help you install Nagios, either from source or via a package manager.

Chapter 4, "Configuring Nagios," is the dreaded configuration chapter. Configuring Nagios for the first time is not something most people consider to be fun, but I hope I've kept it as painless as possible by taking a bottom-up approach, documenting only the most used and required directives, providing up front examples, and specifying exactly what objects refer to what other objects and how.

Most people who try Nagios become attached to it[2] and are loathe to use anything else. But if there is a universal complaint, it is certainly configuration. Chapter 5, "Bootstrapping the Nagios Config Files," takes a bit of a digression to document some of the tools available to make configuration easier to stomach. These include automated discovery tools, as well as graphical user interfaces.

In Chapter 6, "Watching: Monitoring Through the Nagios Plug-ins," we are finally ready to get into the nitty-gritty of watching systems, including specific examples with Nagios plug-in configuration syntax solving real-world problems. I begin with a section on watching Microsoft Windows boxes, followed by a section on UNIX, and ending with the "other stuff" section, which encompasses networking gear and environmental sensors.

Chapter 7, "Scaling Nagios," is new to the second edition. Scaling Nagios for large networks has been one of the most interesting problems Nagios sysadmins have had to deal with over the past five or six years. The explosion of machine virtualization and cost-effective cloud services have created a lot of interest in large parallel processing architectures that are composed of lots of little nodes. In this chapter, I cover several tools and strategies that will enable you to distribute the monitoring load and build a stable large-scale monitoring infrastructure for tens of thousands of nodes and beyond.

Chapter 8, "Visualization," covers one of my favorite topics: data visualization. Good data visualization solves problems that couldn't be solved otherwise, and I'm excited about the options that exist now, as well as what's on the horizon. With fantastic visualization tools like RRDTool, Ganglia, and Graphite, graphing time series data from Nagios is getting easier every day, but this chapter doesn't stop at mere line graphs.

Also new in the second edition is Chapter 9, "Nagios XI," which is dedicated to the new commercial version of Nagios. Built from many of the tools covered in this book by the guys who originally wrote Nagios, XI is truly a masterpiece of integration and usability. They've made monitoring with Nagios so simple my mom could do it (well, my mom writes optimizing cross-compilers for embedded FLIR systems, but you get my point).

And finally, now that you know the rules, it's time to teach you how to break them. At the time of this writing, Chapter 10, "The Nagios Event Broker Interface," is the only print documentation I'm aware of that covers the new Nagios Event Broker interface. The event broker is the most powerful Nagios interface available. Mastering it rewards you with nothing less than the ability to rewrite Chapter 2 for yourself by fundamentally changing any aspect of how Nagios operates or extending it to meet any need you might have. I describe how the event broker works and walk you through building an NEB module.

Who Should Read This Book?

If you are a systems administrator with a closet full of UNIX and Windows systems and assorted network gadgetry, and you need a world-class monitoring system on the cheap, this book is for you. Contrary to what you might expect, building monitoring systems is not a trivial undertaking. Constructing the system that potentially interacts with every TCP-based device in your environment requires a bit of knowledge on your part. But don't let that give you pause; systems monitoring has taught me more than anything else I've done in my career and, in my experience, no matter what your level of knowledge, working with monitoring systems has a tendency to constantly challenge your assumptions, deepen your understanding, and keep you right on the edge of what you know.

To get the most out of this book, you should have a pretty good handle on the text-based Internet protocols that you use regularly, such as SMTP and HTTP. Although it interacts with Windows servers very well, the Nagios daemon is meant to run on Linux, which makes the text pretty Linux heavy, so a passing familiarity with Linux or POSIX-ish systems is helpful. Although not strictly required, you should also have some programming skills. The book has a fair number of code listings, but I've tried to keep them as straightforward and as easy-to-follow as possible. With the exception of Chapter 8, which is exclusively C, the code listings are written in either UNIX shell or Perl.

Perhaps the only strict requirement is that you approach the subject matter with a healthy dose of open curiosity. If something seems unclear, don't be discouraged; check out the online documentation, ask on the lists, or even shoot me an email; I'd be glad to help if I can.

Have fun!
—Dave

End Notes

[1] www.portknocking.org

[2] Dare I say, love it?

This page intentionally left blank

CHAPTER 9

Nagios XI

In 2009, Nagios Enterprises, the corporation formed by Nagios creator Ethan Galstad, launched Nagios XI, a commercial version of Nagios. XI truly is an amazing accomplishment. You need to know next to nothing to use it, and yet the first eight chapters of this book are prerequisite to your understanding it. But now that you have a good handle on how Nagios and the various add-ons surrounding it work, we can finally examine XI and see if it might be a good fit for you.

What Is It?

After the release of 3.0, Nagios was, it seemed, in danger of becoming a victim of its own success. Sysadmins who knew and loved it were happy to see it continue in the way it always had, but its popularity had risen to the point that a different and more populous group of potential end users had taken notice, and with them, Nagios wasn't comparing favorably with newer, prettier, and less flexible commercial competitors.

This new breed of user was quite vocal and had a few very specific gripes. First they found Nagios's configuration syntax unwieldy, to say nothing of the intolerable notion of (gasp) editing text files by hand. Second, they found the Nagios web interface, with its C-based CGI and lack of integrated time-series data, unforgivably old-fashioned. Finally they had no idea what to make of the fact that there was no database back-end. Jiminy Christmas—wrist watches and garbage disposals run MySQL these days! How was one to take seriously a monitoring system that didn't?

For this considerable subset of users, Nagios's price tag didn't make up for its abhorrent lack of bling, and answers to the effect that all these things could be rectified with add-ons fell on deaf Bluetooth earpieces. Add-on options were birds in the bush, and they would rather pay for a bird in the hand than go beating around the bush themselves for free.

XI might best be called the perfect compromise between maintaining the power and flexibility of Nagios and providing a turnkey monitoring system that more than satiates the desires of the PHP proletariat. But that description sells it short; XI is much more than just a shiny interface; it represents a huge amount of custom development and integration work. Further, there is real functionality in XI that simply can't be found in Nagios Core. But neither can it be called a new monitoring system in its own right, because in very important ways, it remains Nagios and retains all the flexibility and power that I've described in the previous chapters. Everything I've written up to now about the underlying architecture, plug-ins, scalability, and even advanced visualization, is applicable to Nagios XI.

It'll be easier to just show you.

How Does It Work?

Figure 9.1 is a rough sketch of the Nagios XI architecture. As you can see, all host and service monitoring, as well as notification, escalation, and so on, relies on an unmodified Nagios Core daemon, so any preexisting plug-ins or customization you might have can be made to work under XI. The NDOUtils plug-in (described in Chapter 7, "Scaling Nagios") has been enabled and configured to replicate state information from Nagios Core into a MySQL database. Here is the primary information hand-off between Core and XI; Nagios XI reads this database to glean information about the current state of hosts and services, as well as the Core daemon itself. This adds an information layer to Core that can be consumed by third-party UIs as well as your own custom integration scripts.

The NagiosQL add-on (described in Chapter 5, "Bootstrapping the Nagios Config Files") provides the hooks necessary to modify the Nagios Core configuration from the XI interface. Every parameter that can be configured in the flat files may be set via the web interface using the customized NagiosQL forms in the "Advanced Configuration" section of the XI interface. Although these forms are well integrated into XI, and retain an XI look and feel, there is a bit of a line in the sand between NagiosQL-driven core configuration, which is referred to as "advanced" in the XI interface, and the configuration parameters that are specific to XI itself.

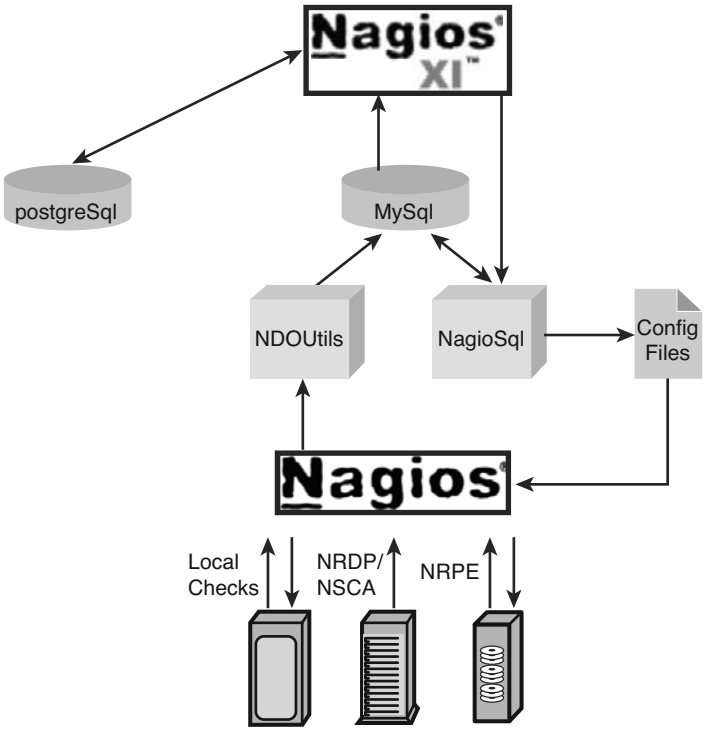

Figure 9.1 The Nagios XI architecture, simplified

XI goes beyond presenting a simple web wrapper to the Nagios Core configuration files, providing in addition a litany of semiautomated wizards and autodiscovery tools to ease the burden of initial and ongoing host and service configuration. I talk more about these later, but suffice to say that it is the intention of the XI creators to isolate the majority of XI users from the intricacies of the Nagios core configuration to the extent that they never need to know what a check command is, much less a template. This makes it possible for monitoring configuration, traditionally an operations task, to be delegated to first-level support types, or in some environments, even to normal users. More clueful administrators who need to customize this or that can still do so, without editing the config files by hand, using the NagiosQL-driven advanced configuration tool.

Configuration created by NagiosQL is automatically written to text configuration files in etc/nagios and is read by the Core daemon from these flat files in the usual fashion. Although it's technically possible to hand edit these configuration files, you will gain nothing

because NagiosQL will eventually overwrite any changes you make. If you have your own configuration-generating automation (like Check_MK), or preexisting configuration that you do not want to import into NagiosQL, or even if you're a curmudgeon who just prefers to manually edit the configuration, you can still maintain static config files in etc/nagios/static, and your files will still be parsed by the Core daemon while being left alone by NagiosQL. That runs both ways; statically configured hosts and services can't be modified via the UI unless you manually import them into NagiosQL (at which point they cease to be static).

Finally, Nagios XI maintains its own Postgresql database to store various configuration parameters such as user-settings, custom dashboards, authentication info, and the like. Given the shiny new PHP interface, the simplified configuration options, and the open database back-end, Nagios XI should satisfy the complaints I'm used to hearing from corporate administrators who are in the market for a "grown up" commercial monitoring product; however, there's a lot more functionality than what I've encompassed in the architecture diagram.

What's in It for Me?

Now that we've taken a quick look at what XI is and how it works, let's take a look at how XI compares to Nagios Core and the various commercial monitoring systems with which it was designed to compete.

One Slick Interface

Given the general quality of the alternative PHP interfaces we find in the Nagios Exchange repository, the XI interface is shockingly excellent. It is certainly not yet another effort to bring the CGI interface "up to date" by replacing it with a PHP version of itself. The XI user interface is a complete rethinking of the UI, which truly takes advantage of the strengths of a web programming platform like PHP at every opportunity. Elements within dashboards can be unlocked, moved around, or even deleted to suit the preferences of the user. AJAX is employed, both to update individual information elements and to provide feedback, so that when I send a command via the UI to reschedule a service check or acknowledge an alert, a box momentarily appears to let me know my command has been accepted. One of my least favorite things about the Core UI is the way it dumps me to an acknowledgment page after I've issued a command, forcing me to manually navigate back to somewhere useful.

The traditional Nagios tables like "service detail" and "hostgroup grid" still exist, but are implemented as repurposable widgets that I can use to build custom dashboards. New tables have been added, a few of which are very dense and handy, like the "minemap" visualization pictured in Figure 9.2.

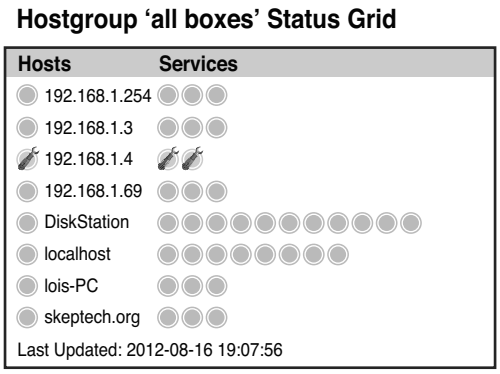

Figure 9.2 Nagios XI Minemap

One of my favorite Nagios Core views is the hostgroup grid, which shows at a glance the state of entire hostgroups, including their services. This is one of the more dense status visualizations available in the old UI; unfortunately, I still need to scroll around to see everything in my environment. The Minemap visualization, by comparison, shows the same information in a much smaller amount of screen space, enabling me to get a coherent, uncluttered, detailed service-level visualization of my entire network on a single screen.

Integrated Time Series Data

PNP4Nagios (described in Chapter 8, "Visualization") is integrated out of the box, and definitions exist for all the included plug-ins. This means that without any additional configuration whatsoever you get time series data for every service you configure. The RRDTool graphs are so well integrated into the UI that the uninitiated user would never guess PNP or RRDTool were community-sourced add-ons, so you get a snazzy UI without losing any of the power and flexibility that these community-driven development efforts provide.

In addition to the RRDTool graphs, small bar-graph visualizations for metrics collected by the Nagios Core daemon, as well as remote execution tools like NRPE, are sprinkled throughout the interface. These do a great job of conveying capacity planning info at a glance, as well as giving the UI a very polished look.

Rounding out the time series visualization is a Graph Explorer tool, which allows you to draw, among other things, ad hoc time series and stacked time series graphs. The graph explorer uses a commercial JavaScript library from HiCharts.com and looks quite elegant. The data comes from the RRD's resident on the Nagios server via rrdtool fetch and is provided to the end-user's browser to compute the graph locally. This saves the server's

CPU and provides a snappy, feature-rich data visualization, allowing you to scale the graph by dragging to select a range and providing pop-up numerical values when you mouse over any data areas. The stacked time series graphs include time-shifted historical data, so you can easily compare today's data to that of yesterday, and so on.

Modularized Components

The UI as a whole is highly modular, incorporating add-on components to implement extra features. This enables the XI developers to quickly react to the needs of the user community by adding features to the UI as needed or even adding custom developing features for larger end users with special needs. A notable example is the Operations screen depicted in Figure 9.3, which is intended to be displayed on a dedicated screen in a Network Operations Center. In addition to this and other single-page summaries, custom views can be configured to rotate between pages with more detailed information on timed intervals. I bring up these little summary views because seeing them so prominently displayed in the XI interface hits home for both the extent to which the Nagios developers are listening to the needs of the community and their eagerness to satisfy those needs now that incremental progress in the UI is possible.

Figure 9.3 Nagios XI Operations screen

Finally! Acknowledgments and Scheduled Downtime for Multiple Hostsv

Another component that implements a feature for which the core community has been begging for years is the Mass Acknowledgment Component. This allows an admin to schedule downtime and acknowledge problems for groups of hosts and services. I know more than one sysadmin who would purchase XI for this feature alone.

Enhanced Reporting and Advanced Visualization

The XI developers are not solely focused on the community, however, as a quick glance at the Reporting tab in XI shows; they are proactively exploring some interesting data visualization

techniques from the neoformix data-visualization field. Components that implement heat maps, force directed graphs, and stream graphs, as depicted in Figure 9.4, have been added to the classic reporting options. Several shiny new implementations of the core reports are also provided, each of which I find generally cleaner than their legacy counterparts and more likely to impress the wearers of neckties and high heels in our lives. The new reports may be exported in CSV and PDF formats with the click of a button. The button, which links to a predictable URL, makes it possible for the shorts and t-shirt wearers among us to automatically grab the reports with tools like curl and wget.

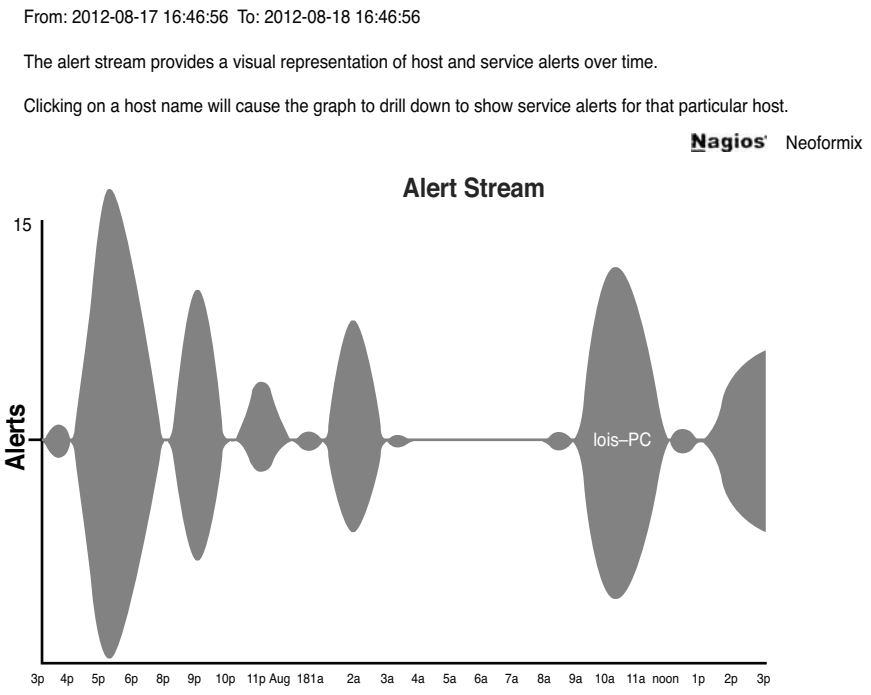

Figure 9.4 Nagios XI Stream Graph component

Nagvis

Nagvis, (described in Chapter 8) is installed and available in the Maps section of the Home view. Setting up your own NagVis diagrams couldn't be easier. First, copy your map or diagram graphic to /usr/local/nagvis/share/userfiles/images/maps, launch the Nagvis tool in the XI UI, select Manage Maps from the options menu, and create a new map, pointing the Background to the map you uploaded. Finally, open your map using the Open menu, and add status icons to it by selecting Add Icon from the Map menu.

Business Processes

Nagios XI contains wrapper logic for grouping individual services into higher-level entities called business processes. The intent here is to implement what the Gardiner Group calls Business Application Monitoring, or BAM. BAM attempts to provide real-time status for critical business entities like a sales catalog web site or corporate email. Nagios XI implements BAM by breaking a high-level concept like "corporate email," into its requisite pieces, such as Mail Transfer Agents, Mail Exchangers, Groupware systems, and Databases, and then quantifying the relative importance of each of the services that make up those pieces as well as describing dependency relationships between them.

XI Business Process groups contain services that are said to be "essential" or "non-essential." A database service in our example might be considered essential, whereas the SMTP port on a single mail exchanger might be "non-essential" (because they are usually redundant, and even if they go down, the mail will queue somewhere else). When any essential service or the combination of all non-essential services goes critical, the XI business process logic registers this as a "problem."

Each business process group contains critical and warning thresholds that depend on the number of problems that are occurring in the group. In our example, we might imagine two business process groups, one for SMTP speakers (MXs and MTAs) and one for SQL-speakers (groupware systems and DBs). If the latter group registers a single problem because a database is down, that might throw the whole group into a warning state.

Business process groups can contain other nested business process groups, and so on. Our top-level entity, corporate email, is therefore just a business process group that contains the two groups previously described. It is configured like the other two groups so that a single "problem" in any of the nested groups causes it to go into a warning state. Finally, notification commands can be assigned on each business process group in the same way they are assigned to individual host and service events. Additionally, visualization widgets exist for the top-level groups. These can be added to any dashboard or view, and they allow the user to drill down into the groups to see what services or subgroups constitute them.

Integrated Plug-ins and Configuration Wizards

The core installation of Nagios XI includes all the plug-ins in the standard plug-ins package, as well as NRPE, NSCA, and NRDP. In addition to all the plug-ins being preinstalled, the XI developers have provided a plethora of semiautomated configuration wizards, which, given the bare-minimum information about a host, take care of the initial setup as well as adding and modifying services on already configured hosts.

If you consult the official XI documentation at

```
http://library.nagios.com/library/products/nagiosxi/documentation,
```

you'll quickly discover that the wizards are the preferred method for host and service configuration. With names like Exchange Server, website, and Windows Workstation, they make setting up new hosts and services easy enough that these tasks can be delegated to first-level support techs, or even end users. The autodiscovery wizard is capable of bootstrapping an environment given only a CIDR netblock to start with, and it does a good job of initial setup. To add NRPE-based host checks or other services after the fact, run the appropriate wizard on the preexisting host.

For example, if Server1 was created with the autodiscovery wizard, and you now want to add NRPE checks to get CPU, memory, and disk information from the host, you must first install NRPE on Server1. If Server1 doesn't already have NRPE on it, and is one of several common server types, such as a Windows 200X server, Red Hat, or Ubuntu, the XI developers have an agent package designed to work with XI specifically at:

```
http://assets.nagios.com/downloads/nagiosxi/wizards
```

After the agent is installed on Server1, run the NRPE Wizard on the server from the configuration tab of the XI user interface, as shown in Figure 9.5, entering the IP or FQDN of the server, and choosing the type from the drop-down list. The wizard will then display a preconfigured subset of available check commands relevant to your server type, and provide text-entry fields for you to specify custom settings or additional commands if you'd like.

As I said earlier, static configuration files may still be maintained in etc/nagios/static. So it's entirely possible to run your own scripts, or autogeneration tools like those included with check_mk, provided you configure them to write their configuration to the static directory. I can't deny that the automated configuration features in XI have, perhaps ironically, complicated things a bit for those of us who have reason to maintain the configuration manually. In the Nagios Core universe, there is a single way to configure Nagios (text files). However, there are three ways to configure Nagios Core in the XI universe (text files, NagiosQL, and XI Wizards), and although the three coexist well enough, it can become burdensome to ensure a uniformity of parameters if the administrators mix and match their configuration methodologies in XI. I'll give you an example.

Larry, his brother Darryl, and his other brother Darryl all work at Bloody Stump Lumber Mill, where they recently purchased a Nagios XI server to monitor their growing sales web-application server farm. Larry was a UNIX admin in college, so he prefers to edit the config files. Darryl likes to have fine-grained control over the config, but isn't very good

in vim, so he uses the XI advanced configuration section, and other Darryl would rather be watching football, so he just runs the wizard for everything. Each of the brothers has a server running sshd that he wants to configure in XI.

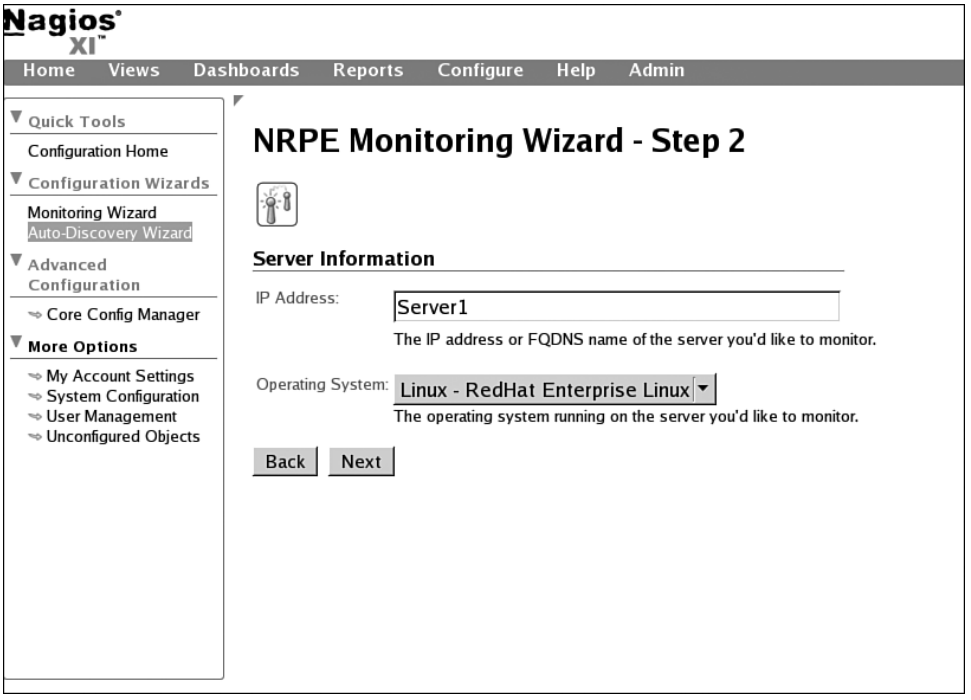

Figure 9.5 The Nagios XI NRPE Wizard

When other Darryl runs the Autodiscovery Wizard on his server's IP, XI scans the host and automatically configures a host check and a check_tcp service check for the SSH port. It then pushes the config to NagiosQL, which commits it to the DB, writes out the configuration, and restarts the daemon.

Darryl meanwhile, sets up his host using the NagiosQL forms directly, but instead of choosing check_tcp, he chooses the check_ssh service, which does pretty much the same thing, but returns slightly different output. He also names the service "ssh" instead of "SSH" like the wizard does.

Larry, meanwhile, has really done his homework. He already has a servicegroup for ssh servers in the static config files he created, so rather than doing all the typing and clicking that his brothers do, he simply adds his server to the ssh_servers servicegroup, and the rest

takes care of itself. The problem is, his servicegroup inherits a different set of templates than NagiosQL, so although his service check uses the same name and check command as the wizard, his polling interval is different, and he has a different notification target for service warnings.

In this way, the brothers end up with three different definitions for the same service, which might not be a problem immediately, but will cause all manner of headaches if and when they want to integrate Nagios with another tool, or generally try to do any sort of automation using their monitoring server.

I admit these sorts of disconnects are possible with text configuration files, but my point is the text configuration encourages administrators to use templates to normalize the configuration, like Larry did in the previous example. The automated tools by comparison encourage isolating the configuration at the host level, because it's easier for the automated tools to parse them that way. Thus, in Larry's configuration, we find a single services.cfg wherein every service is defined and assigned a hostgroup, whereas in NagiosQL's configuration, we find a services directory with a single file for each host. The former makes it pretty easy to verify that all the service checks for every host are implemented in the same way. The later makes it much more difficult.

Further, in my experience, the disdain that people like Larry naturally feel for people like other Darryl generally discourages them from paying close attention to what people like other Darryl are doing. In fact, merely inviting other Darryl to configure the monitoring server with wizards might trigger a tendency in Larry to go off on his own and "do it the right way" using well-written static config files, which only exacerbates the problem by more widely diverging the configuration paths.

Whether this will be a problem in your shop will depend on how many hands are stirring the pot and the extent to which the more clueful users are aware of the potential problem. The idea of delegating the configs is certainly tempting, and I'm not saying you shouldn't. If you do, my advice would be to use either the wizards or static config for service and host *creation*, and avoid using NagiosQL directly if you can avoid it (you could still safely use it for host and service *modification*). That way, you can carefully set up the static config to ensure that it references the wizard templates, or simply copy definitions from the NagiosQL files, and everything should remain pretty much uniform.

Automated Configuration for Passive Checks

Another very cool bit of functionality that is related to automated configuration in Nagios XI is the Unconfigured Objects feature. In the event that XI receives a passive check result for a host or service that it doesn't know about, it automatically generates an inert configuration for that host or service and places it in the Unconfigured Objects section of the Configure tab.

Administrators may then approve the inert objects, and they will become part of the running configuration. Good stuff.

Operational Improvements

In addition to the myriad functional improvements in Nagios XI, several maintenance-related features exist that make it easier to manage the Nagios server itself.

Backups

Out of the box, XI takes a snapshot of the running configuration each time it changes. These configuration snapshots can be downloaded from the UI in an automated fashion using tools like curl or wget. It can be used to restore the configuration in the event the monitoring system kicks the bucket, or it can roll it back to a prior version if someone made an inappropriate change. A real system backup, including historical state and metric data, involves a lot more than just the configuration files, however. Remember, XI maintains three databases and has untold amounts of performance data stored in RRDs, not to mention the Nagios Core state file and logs. For detailed instructions on properly backing up your XI install, see:

```
http://assets.nagios.com/downloads/nagiosxi/docs/Backing_Up_And_
Restoring_XI.pdf
```

User Management

Account management is more important in XI, especially when individual users are encouraged to change configuration parameters and create new hosts and services. Individual users in XI also have the ability to configure the interface with custom views and dashboards as they see fit. For these reasons, XI must track users in its own database rather than leaving it up to Apache to sort out like the Nagios Core UI does. Account management is well done in XI and generally behaves in a manner that enterprise users expect. Access control exists to prevent individual accounts from making modifications, and components exist to enable XI to use LDAP servers. Nagios has published official documentation on multitenant setups, where, for example, access to a Nagios server hosted by a service provider is shared by multiple customers. This documentation resides at:

```
http://assets.nagios.com/downloads/nagiosxi/docs/XI_Multi-Tenancy.pdf
```

Daemon Status

As depicted in Figure 9.6, the XI interface provides an array of detailed of information about the Core daemon process. This includes metric values for the server hardware as well

as performance metrics internal to the daemon itself. A real-time graph of the event queue displays reaper and service check events scheduled 5 minutes into the future. This really is fantastic capacity planning info of a quality I've never seen in any monitoring system.

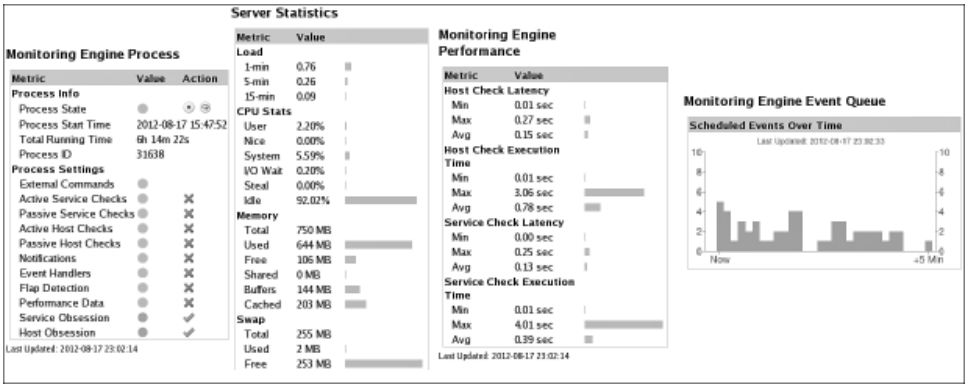

Figure 9.6 Detailed daemon statistics

How Do I Get My Hands on It?

Fully functional demo versions of XI (60-day expiration) are available from nagios.org. You may download self-contained installers, or VMware disk images with XI preinstalled. The latter can be run by any system that supports the free vmplayer utility, while the former requires a relatively recent Red Hat or CentOS install.

The reason for the RHEL dependency is the dizzying array of packages that must exist for XI to run. The XI developers have chosen to rely heavily on YUM to satisfy the requisite dependencies, so although it's possible to run XI on other distros, you won't be able to use the official installation script to get it up and running on anything other than a Red Hat or CentOS system.

This page intentionally left blank

INDEX

Symbols

A

B

C

D

E

F

G

H

listings

M

N

O

P

Q

R

S

T

U

V

W-Z