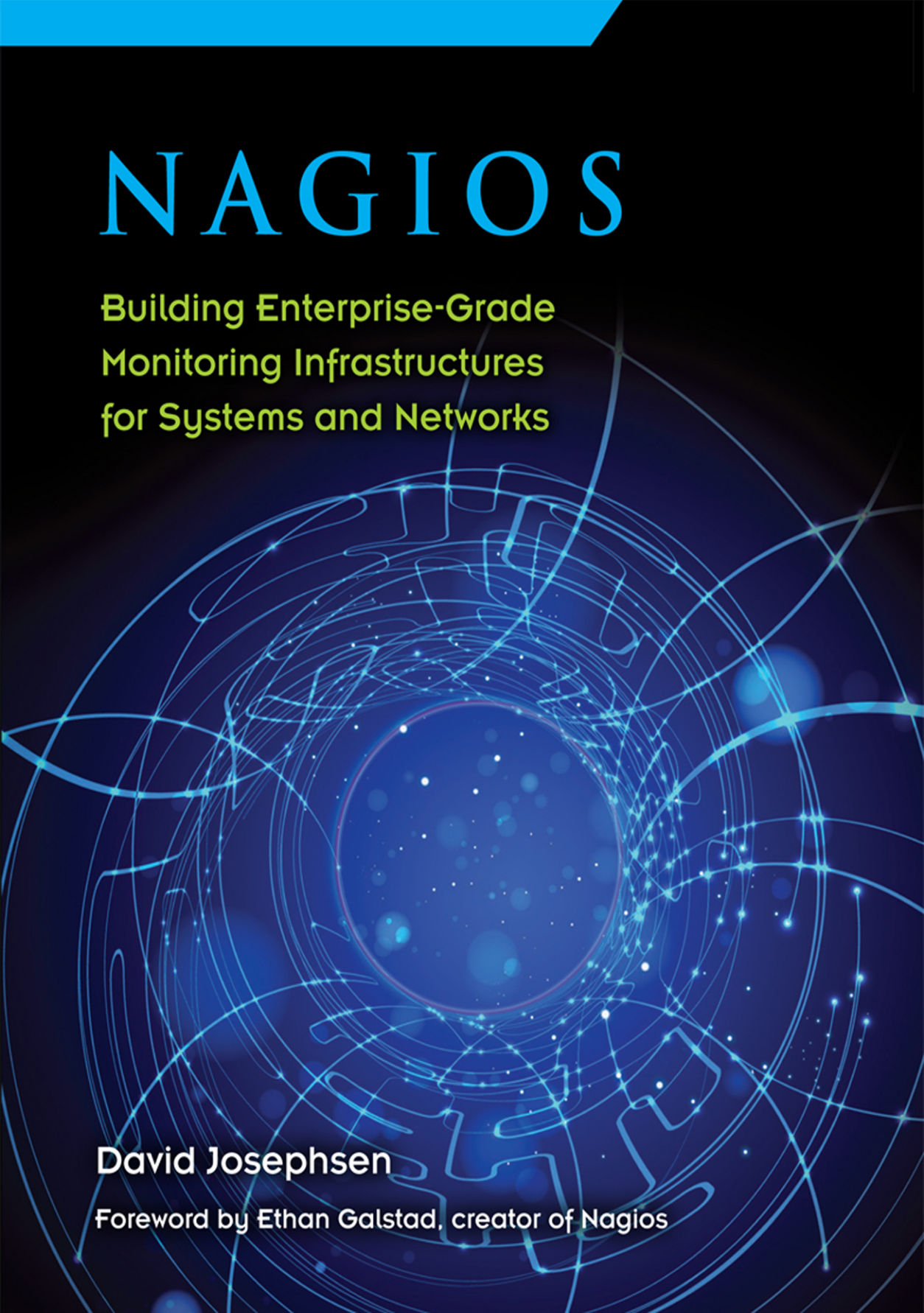
NAGIOS

Building Enterprise-Grade Monitoring Infrastructures for Systems and Networks



David Josephsen

Foreword by Ethan Galstad, creator of Nagios

Nagios

This page intentionally left blank

Nagios

Building Enterprise-Grade Monitoring Infrastructure for Systems and Networks

Second Edition

David Josephsen

Pearson

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 1

Best Practices

Building a monitoring infrastructure is a complex undertaking. The monitoring system can potentially interact with every other system in the environment, and its consumers range from the layman to the highly technical. Building the monitoring infrastructure well requires not just technical aptitude, but also careful planning, a global perspective, and good people skills.

Perhaps most important, building monitoring systems also requires a light touch. An important distinction between good monitoring systems and bad ones is the amount of impact they have on network and server utilization, security, and the people entrusted with keeping things running. "Primum non nocere" isn't just for doctors; if you're building a monitoring infrastructure, it applies to you, too!

The first chapter in this book contains a collection of advice, gleaned from mailing lists such as the nagios-users list, other sysadmins, and hard-won experience. My hope is that this chapter will help you make some important design decisions up front, avoid some common pitfalls, and ensure that the monitoring system you build becomes a huge asset instead of a huge burden.

A Procedural Approach to Systems Monitoring

Good monitoring systems are not stacked up like a house of cards one script at a time by admin in separate silos. They are methodically created by admin with the support of their management and a clear understanding of the environment—both procedural and computational—within which they operate.