iVoyeur
Nagios XI

DAVE JOSEPHSEN

Dave Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.
dave-usenix@skeptech.org

I once had an excellent conversation with Puppet [1] creator Luke Kanies. Well, I think it was excellent but I don't remember exactly. It was LISA '07, I believe, and although I recall talking at length, all that remains in my memory is a vague notion that it was an excellent conversation, along with exactly two details. The first thing I remember is that I embarrassed myself in a particularly epic way. The topic was configuration management engines, and I think I made a quip about not knowing what I'd rather do less, write server configuration in XML or learn Ruby.

Luke, who had just wandered up, told me I should totally learn Ruby, because it was awesome. I responded that I'd recently read a series of articles about Ruby in *;login:* magazine [2], and although the author seemed to know what he was talking about, he hadn't managed to inspire in me a desire to run out and learn yet another OOP scripting language.

Luke, who had written those articles, apologized for not inspiring me.

The second thing I remember is his jacket, which did manage to inspire me. It was a really great jacket, and he wore it as if everyone just went around wearing awesome jackets all the time. As if he came from a land where awesome jackets were the most natural thing in the world and, having only recently arrived, hadn't yet caught on that we were—by our very nature—bereft of great jackets. The rest of us, he probably assumed, must have forgotten our awesome jackets at home or something. Seeing it made me want to be the type of person who could wear a jacket like that. More than that, I wanted to be the type of person who was capable of picking out for himself an awesome jacket to wear places, but, alas, I am not that type of person. Fundamentally, I think, it's a form vs. function thing.

Some of us have a talent for function. We buy ThinkPads, code in C and shell, and build low-level tools, or form higher level tools by combining simpler tools. Others excel at form: they buy MacBooks, code in Ruby, and make polished, shiny tools (with lots of library dependencies). A few of us even straddle the border, dabbling here and there and bringing form and function together.

It's a rare admin who, like me, tends toward function while coveting form. I am often confronted by the truth of this when I, knowing full well that a ThinkPad is what I need, talk myself into buying a MacBook. Like the jacket, it just doesn't fit. Meanwhile, I observe that my function-leaning friends are rarely afflicted with my fickleness. My friend Jeremy would run Linux on a ThinkPad if the keys were made of sandpaper and the pointer-nub was an upside-down thumbtack. My

form-leaning friends, for their part, are even less likely to trade their iPhones for something with a (gasp) real keyboard.

Having been far less hung-over than the others in my party, I was in attendance at LISA '11 when Nagios creator Ethan Galstad was awarded the outstanding achievement award. Before he handed Ethan the award, David Blank-Edelman asked for the Nagios users in the room to raise their hands, and the response was easily 90th percentile. I was a little surprised by this because my impression had been that Nagios wasn't faring well with many sysadmins in the "form" crowd.

Don't get me wrong, the sysadmins who knew and loved Nagios 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 doesn't seem to be comparing favorably with newer, prettier, and less flexible commercial competitors. Nagios was always a functional tool—a framework—and for those of us who lean toward function, it's usually enough that it's possible to make something prettier, that the pieces are all lying around.

This new breed of user seems to disagree. They have a few very specific gripes and are pretty vocal about them [3, 4] (and etc.). First, they find Nagios' configuration syntax unwieldy, to say nothing of the intolerable notion of (gasp) editing text files by hand. Second, they find the Nagios Web interface, with its C-based CGI and lack of pretty graphs, unforgivably old-fashioned. Finally, they seem to have no idea what to make of the fact that there is no database back-end. Jiminy Christmas, wrist watches and garbage disposals run MySQL these days! How is one to take seriously a monitoring system that doesn't?

For this considerable subset of users, Nagios' price tag ($0) doesn't make up for its abhorrent lack of bling, and answers to the effect that all of these things can be rectified with add-ons fall on deaf Bluetooth ear-pieces. Add-ons and hacks are birds in the bush, and they would rather pay for a bird in the hand than go beating around the bush themselves for free.

In the past few years, the Nagios people have therefore had an interesting problem to solve if they wanted to remain relevant: how to create a polished product that compares favorably with the likes of Zabbix, Hyperic, Patrol, Openview, etc. without destroying the extreme flexibility that makes Nagios what it is.

Enter Nagios XI

Nagios XI might best be called the perfect compromise between maintaining the power and flexibility of Nagios, while providing a turn-key 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 an enormous quantity 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 on which so many of us have come to rely.

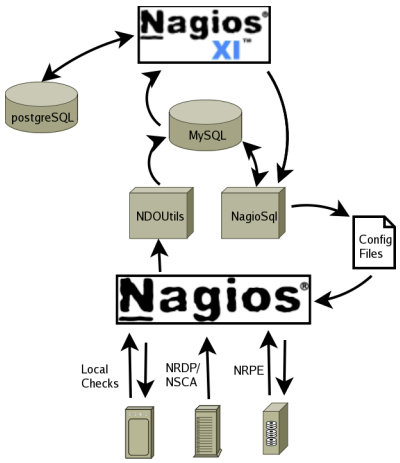

Figure 1: The Nagios XI architecture (simplified)

How Does It Work?

Figure 1 is a rough sketch of the Nagios XI architecture. As you can see, all of the actual host and service monitoring, as well as notification, escalation, etc., relies on an unmodified Nagios Core daemon, so any preexisting plugins or customization you might have will "just work" under XI. The NDOUtils plugin has been enabled and configured to replicate state information from Nagios Core into a MySQL database. Here is the primary information handoff 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.

I've never been a fan of NDOUtils, but this is pretty much exactly the purpose for which it was created: to enable the development of custom Web UIs by giving Web developers (a notoriously database-focused lot) a database from which to glean state data.

NagiosQL, a popular add-on designed to add Web-based configuration to Nagios, 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 instead 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.

This is because XI goes beyond presenting a simple Web wrapper to the Nagios Core configuration files, providing in addition a litany of semi-automated wizards and auto-discovery tools to ease the burden of initial and ongoing host and service configuration. I'll talk more about these in subsequent articles, 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 is exactly what the form-crowd has asked for, and further, it 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, by 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, it will avail you 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 wish to import into NagiosQL, or even if you're a curmudgeon who just prefers to edit the configuration manually, 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 not one but two database back-ends, the shiny new PHP interface, and the simplified configuration options, 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, but again, there's a lot more functionality than what I've encompassed in the architecture diagram.

What's In It for You?

One Slick Interface for starters. Given the general quality of the alternative PHP interfaces I'm used to finding in the Nagios Exchange repository, the XI interface is shockingly excellent. It is not yet another effort to bring the CGI interface "up to date" by replacing it with a PHP version of itself, but a complete rethinking of how the UI should work. It manages to take advantage of the strengths of a Web programming platform like PHP in a way that, like the jacket, awakens my repressed covetousness for style but, unlike the jacket, might actually fit.

Elements within dashboards can be moved around or 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 manually to navigate back to somewhere useful.

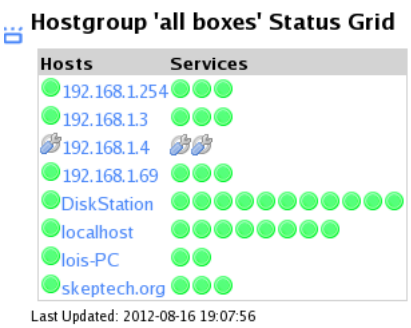

Figure 2: Nagios XI minemap

Traditional Nagios tables like "service detail" and "hostgroup grid" still exist, but are implemented as repurposable widgets, which I can use to build custom dashboards. New tables have been added, a few of which are very dense and handy, such as the "minemap" visualization pictured in Figure 2.

Also included is integrated time series data (pretty graphs), a modularized component architecture with plugins such as the "Mass Acknowledgment Component" (yes acknowledge problems and schedule down time for groups of hosts and services), pretty new reports and interactive JavaScript-based visualization, a means of describing displaying and alerting on business process logic, auto-configuration and configuration wizards, and much more. All of which I will display for you like an awesome jacket in the next few articles.

Take it easy.

References

[1] The Puppet configuration engine: http://docs.puppetlabs.com/.

[2] Luke Kanies, "Why You Should Use Ruby," ;login:, vol. 31, no. 2, April 2006: https://www.usenix.org/publications/login/april-2006-volume-31-number-2/why-you-should-use-ruby.

[3] Nagios grumbling: http://www.frontlineops.net/2011/09/why-im-moving-from-nagios-to-zabbix.html.

[4] More Nagios grumbling: http://blog.desudesudesu.org/?p=1585.