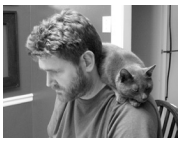
iVoyeur

Changing the Game, Part 4

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

It started with the Vacation. It was only three or four days long, but vacations can be a dangerous time for me. They give me an occasion to pursue subject matter I might normally avoid for fear of the rabbit holes [1]. I'm sure you can relate. One poorly chosen Wikipedia article and a little down-time and suddenly it's 4 a.m. and you find yourself making cheese in the back yard, or thermite in the bathtub. Or was it cheese in the bathtub...? Well you get the point.

Anyway, a bit of heavy reading has lately left me with the palatable sense that my grasp of English grammar is not what it ought to be. That, in fact, it sucks. It's not a happy realization in a person who is paid money to write things.

Realization isn't the right word. I've been aware for some time that my West Coast public school education has left me deficient in this, and many other respects. To be sure, I have the innate grasp of grammar that we all share, but I've never, for example, diagrammed a sentence. Nor have I ever been instructed by a teacher to use the verb to find the subject of a sentence. Before yesterday I was wholly ignorant of how many types of verbs there are (transitive, irregular, dynamic, etc.). Seeing the skill with which these other writers put together words to make sentences to form thoughts, and comparing those sentences and thoughts to my own, has instilled in me a fascination with the rules of language syntax, rules which, if I knew them, would enable me to more accurately and completely (and let us all hope, tersely) articulate my thoughts. Have I been writing in the literary equivalent of Visual Basic my whole life? This is unacceptable.

Now, you and I, being the sort of people we are, the sort of people with training and experience in finding and consuming exactly the right knowledge—not just finding it and consuming it, in fact, but delighting in the finding and consuming of it—we have a penchant for cutting to the heart of things when we put in our cross-hairs subjects like English grammar. You and I aren't surprised to learn that the absolute best way to *understand* English grammar is to learn Latin, and being the sort of people we are, we're comfortable with that in the same way that we're comfortable with the knowledge that the best way to understand Perl is to know assembler.

To those around us, however—those outside the confines of whatever convention hall we happen to be occupying (and perhaps even those to whom we are married)—the idea of learning Latin is a strange, extreme, and probably elitist proposition. That you and I would even consider such an undertaking makes us, transitively, strange, extreme, and probably elitist people. I know this, even if I don't understand it, and so I maintain a cognitive duality to protect the anti-intel-

lectual sensibilities of my fellow man (and spouse). I tell myself I'm not learning Latin, that, in fact, I'm only learning *about* learning Latin, but I hesitate to mention even that to anyone off the conference floor, because that's just the sort of distinction a weirdo extremist would make. No, this undertaking must be a secret. We'll have to keep it between us.

It's vexing therefore, when I've fallen far enough into a rabbit hole that I find myself immersed in the study of "Latin for Mountain Men" [2] in secret, as if it were some kind of weirdo extremist samizdat/porn, to walk into the break room and hear someone announce:

"I have discovered the secret of speed eating. The secret is to make your meal broth heavy!"

Any sort of loudly asserted absurdity like this really shakes me up when I've been on a bit of a mental binge. It makes me feel somehow dissonant and inhuman. I've often suspected that these are the sorts of situations that make people like you and me become people like the Unabomber, so some time ago I developed a mental model to protect my psyche in these sorts of situations. I call it the "Inverse Feynman Filter". I'll let Dick explain:

> I had a scheme . . . when somebody is explaining something that I'm trying to understand: I keep making up examples. For instance, the mathematicians would come in with a terrific theorem, and they're all excited. As they're telling me the conditions of the theorem, I construct something which fits all the conditions. You know, you have a set (one ball)—disjoint (two balls). Then the balls turn colors, grow hairs, or whatever, in my head as they put more conditions on. Finally they state the theorem, which is some dumb thing about the ball which isn't true for my hairy green ball thing, so I say, 'False!' [3]

If a Feynman filter is a mental model for the simplification of complex theorems, my Inverse Feynman Filter is a mental model for the complication of the absurd and idiotic. For example, this speed eating of broth thing sounds to me like a data compression or maybe a signal processing problem. You're taking food, and compressing it to broth, so it can be processed more quickly. See how wonderfully that works? If the subject matter is data compression, we needn't concern ourselves with why someone would want to speed eat, much less that someone felt the necessity to contemplate its secrets. We can ignore entirely the question of whether "heavy" is modifying "meal" or "broth" (why would the weight of the meal-broth matter?) and his improper use of "of" (assuming he meant that he'd discovered the secret *to* speed eating), so bonus, our top secret grammatical endeavors remain undiscovered! We can even interact, observing, for example, that some types of data are more difficult to compress, like so:

"That's fascinating. How exactly does one 'make' one's meal 'broth heavy'? How would I, for example, go about 'speed eating' sunflower seeds?"

"No, no. If you want to speed eat, you have to eat broth."

"So your 'discovery' is really that you can drink soup quickly?"

"Well, yeah, I guess."

As you can see, it's not a perfect model. When it fails, I simply transition to my backup technique, which is to use the story as a lengthy intro to a monitoring column for *;login:*. This way I can focus my mental energy toward the creation of

an appropriate segue from the story into the proper subject matter of the column. The speed eating of broth is, for example, a perfect metaphor for Mathias Kettner's Nagios plugin, Check_MK [4].

Centralized polling engines like Nagios are difficult to scale because the load increase is linear. Every new service on every new host makes the monitoring system work that much harder. Eventually, you'll hit an upper-level limit on the number of services a single poller can handle. Depending on the polling interval—whether you're processing performance data locally, and the sheer horsepower of the hardware on which the poller is running—that limit is generally around 5000 services. At that point we need to look at splitting the workload across multiple pollers. Various means exist to split and parallelize the polling workload, most of which I've described in this column at one time or another.

But what if we could "brothify" some of those service checks, so that instead of performing seven service checks on a host, we could perform a single check on the host that would return the status of all seven services? This isn't a new idea—there have been various attempts to brothify and speed eat service checks over the years—but the idea hasn't caught on, because the implementations were problematic. To be fair, the problem is really the design of Nagios, which assumes that every check returns a singular result from an individual service. The configuration associated with the brothification of multiple service checks is therefore invariably some kludgey mess involving passive service checks which, not unlike brothifying sunflower seeds, is just not worth the effort.

When one considers the inevitable differences between various types of hosts, that some will run services others won't, and that some will use alert thresholds that are more or less strict, it's easy to imagine the configuration nightmare associated with our broth. The specifics of what to monitor and how to monitor it will either need to be moved off the poller and out to the monitored hosts, abdicating the advantages of centralized configuration, or into the check command itself such that the check command for each host is accompanied by three pages of options, only a few of which are actually specific to that host.

Check_MK solves all of these problems and more, providing not just a means to brothify all the service checks on a host, but an all-inclusive monitoring agent that dynamically detects and reports a litany of information about the host. Perhaps "solves" is a strong word, as the server-side configuration is still a mess involving passive service checks, but Check_MK creates and manages all of that for you. In a way, the plugin's dynamic configuration is the most impressive thing about it. After installing the plugin server side, and the agent on the host, the Admin runs an inventory program, which dynamically detects and, through the clever use of Nagios templates, generates the complete server-side configuration for every host inventoried, including the active check for the host as well as the passive checks for each service detected. This was a heavy lift; the kind of programming few of us enjoy.

The agent is tiny, being a shell script running under xinetd on Linux. Unlike NRPE [5], no attributes or arguments are passed from the server, which limits the vulnerability footprint. The agent is easily extended for custom broth ingredients by way of a plugin directory into which the admin may drop his own scripts. These custom scripts will be called by the MK agent, and, assuming they follow some simple formatting rules, their output will be parsed by the server plugin without any addi-

tional configuration. The plugin will in turn generate passive checks for them and report them back to Nagios.

By default the agent dispenses a broth with the big four food groups—CPU, RAM, disk, and network—auto-detecting in the process CPU numbers, NICs (including virtual interfaces like tun/tap) and even disk partitions. A dizzying array of other data is thrown in for flavor, including a process list, and a host of hardware-specific info about devices like NVIDIA and 3-Ware cards, ACPI, and on and on. An agent is even available for Windows which includes all sorts of Windows and Active Directory metrics. A full list can be had by calling the plugin on the command line with a -M switch.

The agent program passes status to the plugin in a way that draws a distinction between mere service state and performance data. The plugin is, in turn, aware of performance data, which it can send to an RRDtool front-end for Nagios called "PNP4Nagios" [6]. The plugin even automatically generates the appropriate action_url syntax in the Nagios configuration so that the performance data graphs generated by PNP4Nagios are displayed on the Nagios Web Interface, all without the admin needing to lift a finger.

The Check_MK plugin provides hooks to customize the configuration it generates, making it easy to specify alert thresholds for individual services on individual hosts. The rules are implemented as a cascading series of defaults, with the most specific match winning. It can also query SNMP devices such as routers and switches using snmpwalk in lieu of a host-side agent.

It's possible that by mixing our service checks together into a broth, we might learn something about how they interact. The Check_MK plugin has a few neat features that explore this possibility, including the ability to detect the primary node in an HA-Cluster using service information returned by the agents, and a feature called "Service aggregations." This latter is an attempt to capture business logic, and bears some explanation because it's actually quite a powerful idea.

A service aggregation can be thought of as a virtual service that is made up of several real services: for example, one can imagine a virtual service called "Email," which is made up of the qmail-send daemon on several hosts along with a few database and HTTP processes on various other hosts throughout the infrastructure. If any of these individual services goes down, Check_MK marks the top-level virtual service down as well.

Check_MK is a well-designed system that should be considered as a replacement for NRPE, especially if you're experiencing growing pains. Next time I'll be covering another Mathias Kettner creation called "MK_Livestatus," which is actually an idea he stole from me about a year before I had it. Until then, look out for the rabbit holes.

Take it easy.

References

[1] "Down the rabbit hole": a metaphor for adventure into the unknown, from its use in *Alice's Adventures in Wonderland*.

[2] Latin for Mountain Men: http://mysite.du.edu/~etuttle/classics/latin/learnlat.htm.

[3] *Surely You're Joking Mr Feynman*, p. 85: http://books.google.com/books?id=7papZR4oVssC&pg=PA85&lpg=PA85.

[4] Check_MK home page: http://mathias-kettner.de/check_mk.html.

[5] NRPE, Nagios Remote Plugin Executor: http://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details.

[6] PNP4Nagios: http://docs.pnp4nagios.org/pnp-0.6/start.