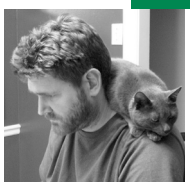
DAVE JOSEPHSEN

iVoyeur: Who invited the salesmen?



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 SIGH INWARDLY AS I TRY TO FIND SOME meaning in the bullets of suit-speak. "Rapid deployment to deliver immediate business value and rapid application development using pre-built components and lightweight scripting!" exclaims the cheerful bullet-point. "Do you promise?" I ask. It doesn't answer, and I continue to the next. With each bullet the words seem to expand and the meaning contract until I begin to suspect an inversely proportional relationship. "Is grammar an elective in techno-sales-babble school?" I wonder. Then I hit the really strange part: "A low cost, open source, subscription model with minimal up-front investment . . .".

So strange and disorienting. Two or three years ago, one could at least count on annoying corporate-ware to be pricey and proprietary. Those of us trying to avoid being stuck with it could shout "Total Cost of Ownership!" and stack the numbers until the costs neared infinity. The open source stuff, on the other hand, used to come with big red warning labels, "DANGER, NO SUPPORT," "MIDDLE-MANAGER BEWARE! CAREER-ENDING CONTENT WITHIN." There were lines in the sand, borders beyond which neither side dared tread. Lately they seem to have disappeared, and in their place have sprouted these weird hybrids.

I vaguely remember believing that open source licenses were going to remove the abstraction created by the "sales" part of the software industry. Companies would choose what they wanted to use based on the geeky details, and then simply pay for support if they wanted it. Although for a time things seemed to be working this way, it now seems silly to have expected that the selection process was going to be geek-driven. Now that open-source is moving into the corporate-ware realm, the sales model doesn't seem to have changed much from that of proprietary software.

There's no reason it should have. The distinction between a support license and a software license is moot to the salesmen and managers, so rather than coming in under the radar by targeting geeks, the "corporate" open-source tools have simply hired salesmen and followed their competition in through the front door. For its part, the open-source license has become a powerful sales tool when competing with an entrenched high-dollar

proprietary competitor. Neither the salesmen nor the managers care one iota about actually having the source, but in their own way, they've finally seen the light (did we "win" without noticing, or have we been assimilated?).

Zimbra vs. Exchange, Alfresco vs. SharePoint—these open-source upstarts with their suit-savvy pre-sales teams, "pay if you want" licenses, and geek-friendly underpinnings are selling, and, despite the marketing-speak, they've managed to build some tools I don't mind working with all that much. Some of them anyway . . . at least, compared to the alternatives . . . sorry, these tools are good enough not to need those disclaimers. The fact that I can't help but write them is a deficiency on my part. It's great that an open-source Exchange even exists, let alone one that I'd actually consider using over mutt, but it's also very weird. I've been rolling my eyes for too long to start raising my eyebrows now.

If you've played with more than one of the "corporate" open-source tools, however, you may have noticed that they're not all created equal. Especially from a monitoring standpoint, some of them tend toward being black boxes and others don't, and this, in my experience, depends mainly on the tools from which they're constructed.

Zimbra [1], for example, is really just a glue layer that holds together a myriad of tools most of us are familiar with: Postfix, SpamAssassin, ClamAV, mySQL, Apache, OpenLDAP, etc. To this they've added an AJAX front end and a boatload of Java glue-code. From a monitoring perspective, this is pretty good news. If the designers haven't provided a decent way to monitor the whole, I might at least be able to get my hooks into the parts.

But, more importantly, this observation, that enterprise software can be nothing but glue and a veneer for a gaggle of seemingly unrelated open source tools, is fraught with portent. With all of these free, mature tools lying around, why wouldn't you pick them up and use them like Legos to build something the suits will pay for? It also smacks of good design to me. It's philosophically compatible with UNIX, saves development time, leverages existing geek know-how, and promises to be easier to troubleshoot, debug, and monitor. All of that, of course, assumes that the tools being glued together are themselves transparent.

Eric Raymond once observed that "a truly great tool lends itself to uses you never expected" [2]. I agree, but I also predict that quite a few mediocre tools, ones that don't lend themselves to unexpected uses, will become parts of much larger packaged solutions—packaged solutions that I will eventually have to deal with. Having a whole that is composed of some parts I can monitor and others I can't gives me pause and invites back my eye-rolling wariness. Let me give you an example.

Java sucks—from a monitoring perspective, I mean. The JVM model makes it difficult to monitor, and the more you have going on inside the JVM, the harder it is to figure out what's happening in there. (I wrote an article [3] about it a while ago, in fact.) Worse, the monitoring hooks that are available (JMX, Mbeans, etc.) all depend on a functional JVM to operate. If (when) the JVM fails, in my experience the monitoring stuff is the first to go. Engineers have a phrase to describe this sort of thing (you may be familiar with it): they call it "in-band signaling." Or, as my wife likes to say, "asking the devil how hot it is in hell."

Zimbra already has this problem. Most, if not all, of the glue runs in a JVM. I don't like this for the same reason I don't like the concept of SNMP traps (versus SNMP polling). If you're asking the thing that might break to let you know when it breaks, well, then you're asking for it. With pretty much any-

thing else, I'd have a control channel separate from the data channel; netstat, iostat, top, ps, and strace all give me meaningful output. With the JVM, if I'm not on a system that supports dtrace I'm screwed. Well, strenuously inconvenienced at least. Anyway, if the Zimbra guys were married to the idea of writing their glue with a portable object-oriented language, the monitoring guy in me wishes they had chosen Ruby or Python.

In fact, the monolithic enterprise glueware concept has quite a few worrisome design considerations. Mostly, it tends to amplify the negative repercussions of doing things Ken and Dennis warned us not to. When you build something from open source tools, you inherit all of the bad habits of every project you select as your building blocks. Coming up with examples isn't difficult.

Imagine an "enterprise content management" system that needs to choose a tool to provide the revisioning subsystem. Three of their primary choices will probably end up being CVS, Subversion, and Git. CVS and Git are great choices. They're both small and have good transparent back ends. I can operate on CVS with file system tools if something goes horribly wrong and Git provides shell tools that give me similar capabilities. Neither of them depends on much and they're both very fast. Subversion, on the other hand, is a huge opaque beast with myriad dependencies, but it has cool factor. Not unlike Java, it's what all the cool kids are using, and for this reason it wouldn't surprise me in the least if it beat out Git and CVS to get bundled into a corporate-ware content management system.

The things that Git and CVS do correctly are the old-school "Zen of UNIX" pieces of wisdom that have been drummed into our heads for years. Do one thing, keep things simple, use text protocols, etc. Subversion went quite the opposite route, packing in all manner of non-essential complexity. Subversion wants to be an end-user program. When it becomes the underpinnings of a larger beast, things will get ugly. Interestingly, neither Git nor CVS was designed to be driven by a larger parent program, but they lend themselves to it because they were designed with UNIX sensibilities. It seems like every time we think things are sufficiently advanced that we can afford bloat, interdependencies, and more abstraction, unexpected use cases come along to prove us wrong. If we ever take notice, we seldom seem to care. But I digress.

There are security ramifications as well. Vulnerabilities are at least as easily inherited as bad habits, and having to implement a protocol or two (a famously difficult thing to do correctly) is a likely problem for the glue code to have to tackle. The Zimbra architects were sensitive to this, ensuring that all of the pieces could talk to each other through TLS tunnels, but the devil is always in the details and mistakes are easy to make. Even given a perfect implementation, the admin still has to go the extra mile to enable things like TLS inter-process communication. Further, it's no great leap of the imagination to assume glue code will be written, which intentionally undermines the security model of otherwise innocent open-source tools.

On the other hand, it's certainly arguable that the glue-code model generally enhances security. For example, bundling something like Apache instead of rolling your own Web server buys a lot in the peer-review department. As long as the designers aren't lazy, keep their eyes open, and make some careful implementation decisions, we'll all probably be better off in the long run versus the classic monolithic proprietary model. It's probably a toss-up.

Here's an interesting question: How long will it be before a project that you contribute to becomes re-packaged by open-source corporate-ware that the company you work for might end up using? In other words, how long before

you become an employee of your employer's vendor? If that's at all likely for you, I'd suggest you begin thinking now about how well your project could integrate. It might save you from having to flame yourself later.

Take it easy.

REFERENCES

[1] http://www.zimbra.com.

[2] Eric Raymond, "The Cathedral and the Bazaar": http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s08.html.

[3] David Josephsen, "iVoyeur: Opaque Brews," *;login:*, October 2007: http://www.usenix.org/publications/login/2007-10/pdfs/josephsen.pdf.