Using Open Source for a Profitable Startup

David A.E. Wall, Yozons, Inc.

The viability of open source projects such as Linux and GNU is frequently questioned. This month, David Wall shows how the use of such software can help a small, underfunded company establish itself in the commercial software arena. It's apparent that from Wall's perspective the viability question has been answered with a resounding yes.

Michael Lutz, Area Editor

When money's scarce, open source software can help your new business launch without breaking your budget.

Like so many tech startups, my partners and I founded Yozons with what we believed would be a hugely successful idea: a unique process for incorporating public-key cryptography that any businessperson could easily use. We believed it would revolutionize the document-driven economy, becoming the killer app for electronic signatures and ushering in the paperless office.

Trouble was, we didn't have sufficient capital to build the business. Undeterred, we immediately documented our process and filed a provisional patent, which gave us only one critical year to raise money, develop the technology, raise more money, then submit the final patent application.

WEATHERING AN INVESTMENT DRY SPELL

When the stock market bubble burst and raising capital became a brutal challenge, we realized we'd have to bootstrap our company. Our calculations showed we needed at least $200,000 to build our software on a shoestring budget—which meant using open source software to avoid license fees. We agreed we would keep our source code completely separate from the open source code and use highly portable coding techniques so that we could deploy on "real" gear later.

To fund our venture, we relied on friends, family, and a few angel investors who chipped in until we exceeded our goal, letting us start six months after our idea hatched. To build a viable Web services business, we needed to use tools that would let us scale from a single server to the thousands we'd need to handle our projected loads.

If you've recently priced Solaris or NT, an Oracle database, BEA Weblogic, iPlanet, or RSA Security's crypto toolkits, you know that buying them requires deep pockets. Worse, the expensive base software severely limited our ability to sell a reasonably priced Web service and forced us to dramatically increase our customers' licensing costs. It's harder to make money when a moderately priced system costs $200,000.

We turned to open source to save us. I'd acquired some background in it by dabbling in the Linux, GNU, and Apache projects since about 1995. But everything beyond that involved a cycle of research, download, build, test, and compare.

POWERED BY JAVA

My significant experience with Java technologies prompted me to build our system with the highly portable Java language, which can build new systems rapidly. Fortunately, Java can be downloaded freely for Windows, Linux, and Solaris. Next, I decided to use Sun's JDK 1.3 and VA Linux's Intel-based servers running their version of RedHat Linux. Unfortunately, choosing VA Linux stuck me with unsupported hardware because the company recently announced they've abandoned their roots to become a software-only company.

For the Web server, I chose Apache because it is stable, robust, secure, and well known. Better still, our service requires high levels of security, with SSL a must, and OpenSSL has a fine implementation that plugs into Apache using ModSSL. The build was straightforward, incurring no costs beyond the hardware and the production servers' digital certificates.

Web services require dynamic Web pages, however, so I chose an application server next. Fortunately, the Apache project came to the rescue on several Java fronts. First, it offers Jakarta, which includes Tomcat for our Java ServerPages and servlet environment. It also offers an XML parser and a SOAP implementation that we needed for our remote APIs. With these in place, we could serve up secure, dynamic Web pages.

A DATABASE SQL

Next, I focused on the database. I'd read much about MySQL, an apparently superfast database. They'd just added transaction support, but I fretted that this

new combination might not be ready for prime time.

PostgreSQL came to the rescue. It's a good database with a workable, though not entirely compliant, JDBC interface. Its even performance across queries and updates, transaction and large-binary-object support, good locking rules, and online backups proved critical.

We did experience a couple of minor snafus with PostgreSQL. First, JDBC does not support binary objects well, making it important to run a contributed utility that cleans up orphaned objects. Second, having to run vacuum periodically is a pain. This utility, bundled with PostgreSQL, cleans up holes created by deletes and updates, and generates query optimizer statistics. The company plans to have vacuum run automatically in the future, making the database self-cleaning. Third, the JDBC library doesn't support small binary objects, a limitation we got around by base-64 encoding them for storage in a text field. Finally, time stamps in many programming environments resolve in milliseconds, but we couldn't get JDBC to accurately store anything higher than centiseconds. Normally, that's unlikely to be a problem, but if you use time stamps as part of a digital signature or secure hash, missing bits pose a real problem.

For our application to run on multiple, distributed servers, we needed to reliably transfer messages among them all. We wanted to use a transactional messaging platform with guaranteed delivery, preferably one that implemented the Java Messaging Service APIs.

The solution, while not open source, was at least free. SwiftMQ's federated router architecture provided a highly scalable message routing platform that has proven fast and reliable. With the ability to connect remote routers over SSL, secure messaging becomes possible even across the Internet. The company releases bug fixes and new features periodically and also has a user community that can help with configuration issues.

At this point, we had everything necessary for a scalable, distributed Web services platform—at no cost to the startup and with no added licensing costs for our customers.

Startup Lessons Learned

Despite the fear, uncertainty and doubt cast by Microsoft, open source licenses are not inherently evil. If your software is proprietary, however, it's important that you understand the open source licenses being used. If you don't, follow "safe open source" practices:

- don't incorporate any open source code directly into your code;
- don't modify the source code to fit your proprietary needs; and
- if at all possible, don't even download and compile the source code.

The third practice should prevent you from being tempted to do either of the first two.

Know the licenses

Many open source projects provide binaries and follow a lenient BSD license, but many more use the stricter GPL, so it's important to ensure that you always keep your proprietary code separate and distinct from the open source code. Depending on your revenue model, releasing your code as open source may work fine for you.

If you can't convince others to invest, you probably won't be able to raise money or even sell whatever you do build to customers later on. Prototype development is a must, but don't try to build the real thing until you have the money to get it built, deployed, and marketed.

Balance marketing and engineering

Techies often don't understand marketing and sales, but these activities are incredibly hard to do right. They generally differ 180 degrees from engineering activities, and they easily mean more to a business's success than the development effort itself.

It's also true, however, that the product is key. I saw a well-funded startup fail despite a fantastic marketing and sales staff. Although marketing sold many investors on the idea, the company's technologists were weak. Focused more on the tools they'd like to use than on building the founder's vision, the technologists never finished the project, leaving their great sales team with nothing more to show than slideware.

Make size matter

If you expect to license your software to large corporations, you must allow it to run on their hardware and software stacks, even if doing so is costly to your organization. Although open source solutions may be more than adequate and often run better than their commercial counterparts, don't expect your clients to risk their positions by bucking the status quo.

All open source projects are not the same. Some are too small, some die from lack of continued support, and some suffer from quality problems. Chose the projects that match your needs and that have an active user community. Rest assured that if the open source project does die, you will still have the source code. If you stick with standard APIs, switching to another project or even to a commercial vendor will be much less painful.

ENCRYPTION ESSENTIALS

We needed superior encryption to offer e-signatures and guarantee the privacy of the business documents being routed through the Yozons network. A Java Cryptography Extension implementation from Bouncy Castle provided this capability. It offers not only a wide range of

standard cryptography—including Blowfish, Rijndael, and TripleDES—but also public-key cryptography through its RSA asymmetric ciphers and digital-signature capabilities. It's also the only open source JCE implementation we found that supports X.509 digital certificates.

The open source community gave our startup up-front cost savings, highly reliable and feature-rich software, and excellent support networks.

Finally, we had everything needed to build a secure messaging service with e-signatures on a distributed architecture. But how were we going to get multiple developers at multiple locations to write code without causing lots of pain and lost updates?

DISTRIBUTED DEVELOPMENT

We chose the open source packages CVS, WinCVS, and OpenSSH, which leveraged the open source community's vast experience with distributed development. CVS provides a reliable, distributed file repository for source code and binaries, WinCVS provides an easy-to-use interface for Windows developers, and OpenSSH provides secure terminal and file transfer access for both interactive work and for securing communications to the CVS repository.

Completing a working Web service requires software that's not tied to software development. The open source community again played a significant role in these acquisitions. Yozons uses e-mail heavily, providing alerts and sending return receipts when users read or sign documents. JavaMail provides the programmatic interface, but Sendmail provides the SMTP server that delivers our e-mail. For our domain name server, we stuck with the venerable Bind.

Both Sendmail and Bind have worked flawlessly although both also have a history of security issues. Thus, we decided to run them on their own server to help ensure that our application did not fall victim to an attack on them. The recent Code Red outbreak hurt many sites running Microsoft software, but we escaped harm, showing once again that open source software can save time and money over commercial competitors.

Providing accurate time stamps on messages, documents, and e-signatures is a critical Yozons feature. The network time protocol fit the bill, and timeSync offered an easily configured, automated NTP solution that ensures our clocks remain highly accurate and synchronized.

To secure the Web servers from intrusion, we examined a few firewall appliances, but found their costs prohibitive. Fortunately, Linux's ipchains offers a firewall for inbound and outbound traffic. It also provides network address translation to let us run our application on a private network behind yet another firewall. Tripwire and Snort detect unexpected modifications to files and analyze logs for intrusions, turning off all unneeded Internet services, removing all unneeded user accounts, and ensuring high-quality passwords for the accounts that remain. Coupled with these services, Linux has shown to be very secure and robust.

The open source community gave our startup up-front cost savings, highly reliable and feature-rich software, and excellent support networks that let us offer a reasonably priced secure document delivery and e-signature application to our business clients. Although we're pleased that we can easily run our software on Solaris, iPlanet, Oracle, and BEA for those clients who demand it, we're delighted with the results we've received because of the hard work and dedication of many untold open source contributors. ✷

David A.E. Wall *is a founder and the chief software architect for Yozons Inc., a Seattle-based startup focused on providing e-signature and secure document delivery Web services. Contact him at dwall@yozons.com.*