

The "Null Service"

As customers get more interested in hosted services and ASPs, a lot of product teams are re-conceiving their packaged software as outsourced Internet offerings. The assumptions and infrastructure needed for **hosting** a service, however, are very different from traditional licensed software. Hosted corporate applications need an underlying architectural layer that is missing from internal apps -- but roughly consistent across ASP offerings. I've been calling this the "**null service**."



First some history. Years ago at Sybase, we had lots of discussions about the "null release." This was shorthand for the effort needed to put out a new version of a software product **independent** of the specific changes being made to it. Imagine a one-line fix to Sybase's enterprise middleware suite, which then had to be... ported to 31 operating systems, run through QA, compatibility tested, documented, released, staged for customer upgrades and announced. This might represent 4 months work for 45 people, at a cost of \$2M. The "null release" concept helped balance demands for ever-more-frequent release cycles against the fixed cost of production.

Moving forward a decade to the emergence of hosted services and ASPs, here's our related concept: the "null service." This is the technical infrastructure needed to run **any** online service for business – regardless of its specifics – if the service must be secured, hosted, billed for, and generally available to all of its users. Said another way, there is a common set of systems and procedures that most web-based applications should have.



How does pondering the "null service" help? Early in the planning cycle, it lets a marketing and development team cleave large projects in half – with one group focused on well-understood infrastructure while another concentrates on defining detailed service features. Part of the implementation work can begin right away, in parallel with market analysis.

How About an Example?

Imagine that your company is the leader in outplacement application software. The world's largest corporations license your solution to track successive waves of employee downsizing -- assuring HR executives that seniority rules have been followed, and that no Federal statutes have been violated. (This is fictional, remember?) Corporate customers run your software on



their own systems, however, and pay you \$500,000+ each for the privilege. You believe that a much larger market exists as an Internet service if you can lower the price and skinny down the features. By hosting a lightweight version of your application online, you can reach thousands of mid-sized companies that might need to cut staff some day.

True to form, your software designers want a precise map of service features before they start development. To create it, your marketing team needs extensive customer research to pick the few important features from among your hundreds of configuration options. This requirements-gathering cycle will take six months of customer interviews and design work before the first line of application code is written.

Should We Wait for the Perfect MRD?

Don't wait! Now is a great time to begin laying the groundwork for your "null service." Regardless of which downsizing features you include, the solution will have to have:

- Hot stand-by systems for application and database fail-over
- Account information identifying each customer (and each user)
- Password issuance, plus a procedure for replacing lost passwords
- Security to keep each customer's staff cuts hidden from others
- A billing module to send invoices, enroll new customers, show usage-to-date, and cut off 60-day-past-due accounts
- An uptime commitment or Service Level Agreement (SLA), along with reports to measure actual uptime
- Operational procedures for validating software updates and staging them without customer downtime, plus a way to back them out if needed
- "Thank you for enrolling" emails with instructions and URLs for more help
- Self-help portal for current and prospective users
- Customer support processes (email or phone?) with committed response times
- 60-day reminders in advance of annual renewals
- Marketing e-newsletter with subscribe/unsubscribe mechanism
- Posted privacy policy for your e-newsletter
- ...

You get the idea. There is a lot of work queued up, and much of it could be started today. Decisions about how your application should calculate exit packages (or whatever) don't make any difference to uptime strategies, password management or newsletter opt-in policies.

Many of these underlying hosting technologies will be managed by an Operations team. This is a crack set of network and production engineering folks who keep your service running in the face of power outages, typhoons, locusts, sudden peaks in end user traffic, carrier bankruptcy, and email viruses. [An Operations group doesn't exist in your current "licensed software" model](#), but will be critical to the success of any hosted service. Finding them may take a while.

A big part of the Operations puzzle is the design of people processes: detailed steps to do seemingly simple things. Who will take customer support calls and how quickly will we respond? What happens when a customer wants a

partial refund half-way through a subscription period? How do we verify that a password re-issue request is legitimate?

And then...

Ultimately, these two major pieces -- core functional development and Operations -- will have to come together. For instance, you may decide that each account will have several kinds of users (HR managers, outplacement specialists, and ex-employees), which forces some fiddling with login security to set permissions correctly. This then implies a procedure for customers to give out end user accounts, and so on. Having six extra months to design a foundation will speed your implementation and reduce last-minute panic.

Eventually, the "utility computing" initiatives from HP and IBM may meet application infrastructure dreams like [Jamcracker](#) and Salesforce's [sforce](#). This could give us a complete hosted 'shell' for paid, production-quality Internet services -- like the e-commerce and logistics framework that Amazon provides for [Target](#). For now, though, you should plan on architecting most of this in-house.

Sound Bytes

While you're planning the details of your Internet service, you should also start mapping out the generic infrastructure that you'll inevitably need. Don't wait for perfect application clarity to get your operations design underway.

"Product Bytes" is a monthly newsletter about product strategies for C-level executives. To subscribe, send an email to subscribe@mironov.com. Back issues are archived at www.mironov.com.

All contents © 2003 by Rich Mironov. Product Bytes (TM) 2002. Contact rich@mironov.com or 650.315.7394.