CIO

Print Article Close Window

From: www.cio.com

Cloud App Integration: What's the Best Path? Part 2

– David Taber, CIO

January 21, 2011

Nearly every software vendor now makes claims about how they are cloud based, or cloud ready, or crowd friendly. Unfortunately, in doing so they are blurring the meaning of being a cloud application. Consequently, this article will apply differently to every cloud vendor.

With that disclaimer behind us, one of the distinguishing characteristics of cloud software is the variety of ways it can be integrated. In Part 1 of this article, we discussed the four different layers at which cloud applications can be integrated. Now we look at the other dimension: the integration products and tools that are available for cloud and other Web service integration.

Category 0
The first category of integration products is no product at all: using the cloud application's own platform as the mechanism for integration. This means using the native programming language available in the cloud vendor's business logic or database layer to call out directly to another cloud. It's a simple idea, but can be tough to pull off — and in my experience it works well only in simple point-to-point integration use cases. Using this approach, you're likely to have to develop snippets of integration code at both ends of the cloud-conversation...and likely in different programming languages. The more differences your developers have to contend with — RESTful vs SOAP, XML dialect vs JSON, object tree vs DBMS table — the less likely this is going to be a fun exercise.

Even so, direct integration is a viable option for read only and other use cases. Some integrators swear by it. I only swear at it. If you're trying to do something truly transactional across clouds, dealing with message retries, guaranteed delivery, two-phase commit, and rollback logic can result in a prodigious pain in the pituitary. Did I mention dealing with system maintenance windows?

Category 1
The next category of integration products is the point-to-point connector. These come in two flavors: an on-premises version for use on your own servers, or integration services that reside in the cloud. Pervasive, Boomi (Dell), and other vendors use this approach to connect to a wide range of cloud and on-premises applications at a very reasonable cost.

Typically, these off the shelf, point-to-point connectors provide two-way synchronization of a limited number of fields. In some cases, the connectors have no configuration options at all, but even the most flexible connectors allow changes only to field mapping. Adding custom fields, mapping to custom objects, or having your internal code push data directly through their pipes is typically off the table.

These connectors are sometimes all you need for linking, say, Salesforce.com to QuickBooks online. What they provide is really point-to-point data synchronization, not the general case of integration. Trying to use connectors to link a third or fourth cloud is unlikely to work well, particularly if two connectors need to access the same business object.

When evaluating connectors, make sure to evaluate their documentation and support as there is truly horrific variability of this across vendors. Definitely check their forums or other discussion areas to get a realistic appraisal on these issues.

Category 2
The next approach to integrating across clouds is to use a connector that isn't targeted to any particular application, but to an industry standard connection that the other cloud can understand. The most obvious examples are ODBC and JDBC, but there are also language-level connectors that talk to standard libraries.

The benefit of this approach is the flexibility: your developers can get access to an arbitrary number of remote tables, objects, or methods. In addition, these connectors typically aren't very expensive.

But these are still point-to-point connectors, which means they're not appropriate for integrating several clouds together. Further, they don't provide much in the way of high-level services or object/application context. They're pipes.

Category 3
The final approach is to use an integration server, which can itself be a cloud service (hosted at Amazon or elsewhere). Using an integration server still requires the use of connectors at the end-points, but the connectors are essentially "dumb pipes" with all the intelligence at the hub. Because integration servers provide high level services (such as message brokering, translation, retry logic, logging, and administration) as well as their own programming (or at least scripting) environments, developers have a lot more to work with. Context-sensitive routing, workflow, and compensating transactions can all be built out (and, more important, evolved) without disturbing any of the other cloud applications.

An integration server — whether it's in the cloud or on premises — is the most flexible and adaptable approach. There's no architectural limit to the number of systems you can integrate. Nearly any kind of system (including hardware controllers or other legacy systems) can be integrated. And in most cases, an integration server yields the best possible performance. The downside, of course, is that it's totally "build it yourself." While there will be templates and code samples (and who knows, maybe some documentation), the whole point of an integration server is bespoke construction.

Next week, we'll look at another dimension of integration: security.

David Taber is the author of the new Prentice Hall book, "Salesforce.com Secrets of Success" and is the CEO of SalesLogistix, a certified Salesforce.com consultancy focused on business process improvement through use of CRM systems. SalesLogistix clients are in North America, Europe, Israel, and India, and David has over 25 years experience in high tech, including 10 years at the VP level or above.

Follow everything from CIO.com on Twitter @CIOonline.

© 2010 CXO Media Inc.