

Defining “Multi-Tenancy”

Rob Rennie
Founder and CTO, Webiplex LLC
3/24/2009

PaaS Degrees

In “Many degrees of multi-tenancy”¹ Phil Wainwright does a great job in discussing the ambiguity and debate surrounding the classification and definition of “multi-tenancy”. He refers to first, second, and “lesser” degree classifications of multi-tenant systems in relation to Software as a Service (SaaS). These general classifications are reasonable and helpful, but I as many others, think the topic of multi-tenancy is a more nuanced computer science endeavor which might be outside simple classifications. The alternative views proposed herein I believe better align with historical trends going back as far as mainframes – the first multi-tenant systems.

Judging from the comments by readers at the end Mr. Wainwright’s article, there are some pretty strong feelings from multi-tenant “purists” who go so far as to classify anyone who is not (by the reader’s definition) multi-tenant as “fake-SaaS providers” who “will simply fall behind in keeping their solutions up to date”. Or another reader who says if Oracle does not deliver SaaS in a multi-tenant model (again by the reader’s definition) then “this is a blunder and hence my opinion – Oracle hosting is just ASP on juice!”

While many of these arguments focus on the Platform as a Service (PaaS) vendor perspective of the underlying implementation, understanding the nuances of multi-tenancy claims also helps a data analyst or application builder evaluate a given PaaS vendor. Put simply, does the PaaS have an infrastructure that supports all the degrees, pure or “impure”, available in delivering a multi-tenant SaaS application? You wouldn’t want to be accused of using a “fake” PaaS after all and have to fight esoteric technical battles regarding your PaaS provider of choice.

Database Partitioning Scenario

A scenario I find particularly useful in analyzing what multi-tenancy really means (and does not mean) is the case of database partitioning. Let’s say, according to Mr. Wainwright’s article degree definitions, we start with a first-degree multi-tenant SaaS application. Let us also consider a very successful SaaS application that has thousands of tenants on the system; the database is growing very large. We’ve used Microsoft SQL Server² and in an effort to improve performance, and support more efficient growth, we create a database partition scheme using file groups and a partition function. Let’s say we take the “Tenant ID” (every SaaS has a “Tenant ID” of some kind, which is the unique identifier of the tenant) and

¹ See <http://blogs.zdnet.com/SAAS/?p=533>

² Other DB’s support partitioning – Microsoft SQL Server is used as an example.

modulo divide it by the number of file partitions and the result defines the partition where the tenant’s data goes.

Is this still first-degree multi-tenant? According to the strict definition, there is now something that is “less shared” than was before. The database files are no longer shared amongst users. However, the application code doesn’t know the difference - it’s still a single database - so the question of whether we are first-degree or “pure” really begins to get muddled.

Along the same lines, let’s say instead that we create a separate database, identical in structure, for every five tenants that sign up to share (without their knowledge of course). And let’s also say that the PaaS infrastructure makes sure that all databases are exactly in schema-synchronization, and the connections to those databases are abstracted beneath the actual PaaS infrastructural software code itself. In this scenario, we essentially done the same thing as partitioning, but instead we change which level within the programming stack we do it.

You can easily continue this concept upwards and reach the model of the “first-degree Salesforce.com³” where the partitioning happens at the data center level – which I think (according to Mr. Wainwright’s degrees) actually means it’s *not* first-degree. Even this simple scenario starts to shed some light on the fallacy of the “pure” multi-tenancy debate.

Segmenting for Growth

When databases or systems grow large, segmentation, very simply, is what is ultimately brought to bear on the situation. Whether you segment at the database file level, at the database level, at the server or data center level, what you are usually doing is partitioning an index – creating a scheme to get to the queried data faster. Even if you keep everything as a first-degree multi-tenancy design, you’ll someday have to rely on underlying technologies like cloud virtualization to partition things beneath your system – you’re just not dealing with it directly. Some might argue then that the “degrees of multi-tenancy” are in actuality, a matter of “degrees of perspectives” or more exactly “different perspectives”.

It’s interesting to see how the perception of isolation vs. multi-tenancy vs. segmentation changes according to the constituent within the overall multi-tenant system – from user to developer to IT administrator to DBA:

- *End SaaS application user:* The end user view is actually relatively isolated with no exposure to the segmentation. In the case of social networking Web services like MySpace or Facebook for example, there is clear knowledge there are other tenants, but data visibility is from the perspective of an isolated tenant with some connections to other isolated tenants.
- *PaaS application developer / data or process analyst:* The process analyst view is also an isolated view within the overall multi-tenant system. His or her focus is on building an application using a PaaS platform for isolated end SaaS application users.

³ See Wainwright

- *PaaS-vendor internal software developer*: More likely than not, the actual code written by the PaaS vendor is written as if it was an isolated tenant, without regard to the segmentation, and all calls to the data store are "contexted" by the Tenant ID. Only in situations where inter-tenant communication occurs is code written that might span tenants.
- *PaaS-vendor DBA*: This is where the multi-tenancy view really starts to take focus as decisions about partitioning, indexing, storage, etc. are made. However, even at the database level, segmentation could be separated from first-degree database design albeit they are usually intimately linked conceptually by things like partition functions.
- *PaaS-vendor IT administrator*: Multi-tenancy is also a priority perspective here as multiple servers (virtual or real), clusters and even datacenters are considered.

In addition, another useful scenario to think through relative to multi-tenancy is a hosting company who provides virtualized private servers using something like VMWare. The perspectives of isolation vs. multi-tenancy in this scenario also change depending upon which constituent (e.g. end-user, developer, etc.) within the overall system you talk to.

What it all seems to come back to is there is no such thing as a "pure" first-degree multi-tenant system because at some point, something is going to have to get partitioned. Given this is the case, then the only way the concept of multi-tenancy can exist is if the perspective is also stated. The perspective of multi-tenancy then ultimately only exists at the level of those responsible for segmentation and inter-tenant communication.

Inter-Tenant Communication

I think it is also important, when evaluating a PaaS or SaaS application architecture, to understand the inter-tenant communication model. For example, if we look at any social networking Web site like LinkedIn, or MySpace; these are multi-tenant SaaS applications to be sure and they allow for one tenant to contact another tenant.

The question is how is this inter-tenant communication occurring? I do not know for sure, but I'm assuming the communication is occurring via a table of "messages" within the SaaS database as this would make sense. No sockets or pipes or traditional inter-process communication (IPC) connections are being opened between one tenant and the other to transfer this message. Further, once the two tenants are connected (e.g. "friends") there really only needs to be a single table to represent the relationship.

This model does not scale out in a distributed scenario. If for example one of the tenants in the aforementioned system would like to take their data local and install an on-premise version of their tenant software, all the inter-tenant communication breaks. While this is not likely in the social networking Web site, it is extremely likely and often a requirement of business applications.

It could be argued that if a given PaaS does *not* support traditional models of IPC between tenants, then it can *never* really be scaled out. This does in fact result in the non-intuitive idea of building the likes of socket-based or Web services-based inter-tenant communication protocols when the tenants actually reside in the “same database”. This non-intuitiveness makes it difficult to institute early on in the PaaS architecture, but also makes it a nightmare if it is needed later and the PaaS or SaaS wasn’t originally built that way.

Isn’t (wasn’t) P2P Multi-Tenant?

If it is accepted that in order to scale on-premise and SaaS – scaling out in a massively distributed way – using traditional IPC, then the “peer-to-peer” (P2P) paradigm (a la Napster) as an extreme example of a completely distributed multi-tenant system could be argued. In the Napster P2P case, the perspective of segmentation is actually right in the user’s face every time they do a search for their favorite song.

In the P2P model, there is also a shared schema and there is inter-tenant communication. The inherent segmentation of the system itself is its primary feature and is exposed at all levels of users (perspectives) of the system. Software code is written in the P2P model that is multi-tenant aware at all levels. Many of the requirements of the new “multi-tenant SaaS” systems are met in the old P2P model, it’s just the perspectives of the multi-tenancy that change.

“User Defined Field” Trap and Custom Tables

A common facet of nearly any business software system is the venerable, maligned, and over-abused “user defined field”. Generally speaking, business software is developed around a common business process – say customer relationship management (CRM) for example. The end-users ultimately have the need to record information entirely specific to their business not included within the base business process of the business system. For example, a certain customer wants to add “accounting system id number” to all the “customer” records within their CRM.

To accommodate this, software vendors traditionally add an arbitrary number of “user defined fields” to the end of specific tables within the underlying database. Since the software vendor can’t anticipate the data types of these fields, they usually add it as a text field, or they’ll add multiple user defined fields with different types – like a “user defined integer”. There are even systems⁴ whose entire model of user-customized “forms” is based on unnormalized user defined fields like this.

The problem is these “user defined fields” are not first class residents in the database. They either represent unused space, or if used, they force strange machinations like storing integers in string fields which sort according to alphabetic rules. And they never seem to appear in the user interface where the user ultimately wants them to appear, and do not participate in the finely tuned analysis done by the database developer for the other fields.

The user defined field model can certainly work in the multi-tenant environment too. But what if we want to deliver a multi-tenant platform the lets application designer (who is using the PaaS) build first class database entities? So, when they create a “form” for end-user data entry for example, the fields

⁴ From the some of the biggest software vendors you can think of.

on the form are stored in a table that actually has the right field names and types. Furthermore, if they search on one of those form fields a lot, they can add a database index just for that field. What impact does this have in a multi-tenant environment?

First, we are immediately pushed outside of shared tables so we can no longer be considered a first-degree multi-tenant application – or can we? Does this conversely mean that to be a first-degree multi-tenant application you *cannot* provide end-users the ability to create data definitions resulting in first class database entities? Whether these tables are stored in the “main” database or in a secondary database for custom tables is irrelevant and doesn’t change the question.

Summary – Pick the Right PaaS

It all comes down to choosing a PaaS that you know supports *all* the options, degrees, or models, of multi-tenancy implementation. When you choose the PaaS that supports more models, then you inherently have a PaaS that supports a greater degree of flexibility for your particular application needs. If you choose a PaaS that manages and abstracts segmentation and partitioning at *both* higher⁵ and lower⁶ levels, then you can rest assured you can deliver on even the most demanding performance and customization requirements of your own customers or end-users.

It also means that there really is no such thing as pure first-degree multi-tenant systems without also stating from whose perspective the classification is made. Models for partitioning extremely large data sets efficiently have been around for decades and involve segmentation of some kind. The fact that the primary key is a Web client “tenant” doesn’t change anything. Conversely, business software customers expect *customized* applications which have been delivered for decades. A first-degree SaaS is not viewed as an innovation by a end-user company that loses their customized IT competitive edge.

A true PaaS recognizes these facts and delivers the financial benefits of shared resources and competitive benefits of customization, regardless of the particular flavor of multi-tenant segmentation chosen.

About the Author

Rob Rennie is founder and CTO of Webiplex, makers of DocuPeak. Webiplex DocuPeak is a business process management (BPM) and document management (EDMS) PaaS which is implemented atop a true cloud data center providing unlimited scalability to our customers. Webiplex delivers tools for the business process analyst to use to construct applications and does not require programming or extensive IT knowledge. Our customers include both the public and private sector, and our implementations are both on-premise and hosted.

Prior to Webiplex, Rob was CTO at Loan-Score Decisioning systems, a multi-tenant SaaS delivering automated underwriting solutions to the mortgage industry and before that, Rob was founder of theSupplyChain.com – a multi-tenant SaaS solution created in 2000 for managing inventory, shipping, and ERP transactions.

⁵ “Lesser-degree” from Mr. Wainwright’s article

⁶ “First degree” from the same article

Rob can be reached by contacting Webiplex via <http://www.Webiplex.com/>

Trademarks

All product names mentioned in this document are property of the respective companies that hold those trademarks.