# SaaS Multi-Tenancy: Framework, Technology, and Case Study

*Hong Cai, IBM China Software Development Lab, China*

*Berthold Reinwald, IBM Almaden Research Center, USA*

*Ning Wang, IBM China Software Development Lab, China*

*Chang Jie Guo, IBM China Research Lab, China*

## ABSTRACT

*SaaS (Software as a Service) provides new business opportunities for application providers to serve more customers in a scalable and cost-effective way. SaaS also raises new challenges and one of them is multi-tenancy. Multi-tenancy is the requirement of deploying only one shared application to serve multiple customers (i.e. tenant) instead of deploying one dedicated application for each customer. This paper describes the authors' practice of developing and deploying multi-tenant technologies. This paper targets a technology that could quickly enable existing Java EE (Enterprise Edition) applications to be multi-tenancy enabled thus having the benefit of quick time to market. This paper describes the overall framework of multi-tenant SaaS platform, how to migrate an existing Java EE application, how to provision the multi-tenant application, and how to onboard the tenants. The paper also shows experiments which compare the economics of multi-tenant SaaS deployment versus traditional application deployment (one application for one tenant) with precise data.*

*Keywords:* *Cloud Computing, Java EE, Multi-Tenancy, Programming Model, SaaS*

## 1. INTRODUCTION

In recent years, we have witnessed the evolution of research on Utility Computing, Grid Computing, virtualization technology, SaaS (Wikipedia, n.d.), next generation Web technologies, and Services Computing that all together have yield a new research area named Cloud Computing (Wikipedia, n.d.). Cloud Computing promises to provide easy to use service oriented user experience to allow massive end users to use IT resources and IT applications as services without on-premise installing the IT infrastructures.

A technical definition of Cloud Computing is "a computing capability that provides an abstraction between the computing resource and its underlying technical architecture (e.g., servers, storage, networks), enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction." This definition states that Clouds have five essential characteristics: on-demand

DOI: 10.4018/ijcac.2011010105

self-service, broad network access, resource pooling, rapid elasticity, and measured service.

This definition by default assumes the Cloud services as Infrastructure as a Service (IaaS). IaaS could be very successful in development and testing environment, because those public Cloud offering cheap and instant infrastructure services could ease the reach to IT resources. Amazon EC2 (Amazon Web Services, n.d.) is a well-known and typical public Cloud offering. On the other side, through two years practices in the Cloud market, we have observed that customers in different geographies typically have different market requirements of using Cloud services. For example, in the emerging market there are strong needs of e-commerce, accounting and ERP applications for small and medium business (SMB), so these SMBs can save investment on IT infrastructure and spend a bigger budget on their direct business operations such as marketing and businesses development. With this intent, SaaS is often their first resort. Today, some business applications and software have been developed to a stage that most of the functions are good enough to most of the customers. The ultimate goal of SaaS is to provide software to those segmentations of customers with flexible deployment (without on-premise installation) and pricing model (usually by paying monthly or annually subscription fee). However, SaaS has different maturity levels with varying capabilities and cost structures (Chong & Carraro, 2006). In its basic levels (Level 1/2), lower layer IT infrastructure resources are shared. In Level 3/4, multi-tenancy is the core feature with single instance of application serving multiple customers (tenants) to achieve the minimum operational cost, and maximize the revenue of the SaaS operator.

A key concept in SaaS multi-tenancy is "isolation point". Essentially, isolation points are artifacts or resources that need to be isolated for different tenants in shared Web application instances. An example of isolation point is a visitor counter implemented with static field. Requests to different tenants need to trigger the counters to increase their counts separately. Details of isolation points will be introduced in Section III. To serve multiple tenants, a SaaS application needs to have all its isolation points identified and isolated at runtime. Multi-tenancy is the core technology of SaaS, and we need to understand there are different options to implement multi-tenancy to enable Web applications which have Web UI, business logic, and database. One option of implementing multi-tenancy is taken by Salesforce.com who provides Web based SaaS application development based on a template of the application (such as CRM application). The problem of this option is that the application developers have to be trained to learn the new programming model, and pay more efforts to transforming those legacy applications with the new programming and runtime platform. The limitation is that there's little fine granularity customization exposed such as adding, updating, or deleting a class, a method, or a field. The second option of realizing multi-tenancy is through designing multi-tenancy entry points at design time (Osipov, Goldszmidt, Taylor, & Poddar, 2009) so that different tenants may have different service components through different entry points. This approach will allow the developers to follow the original program model they are familiar. The limit of this option is that the designer and developer of the application need to design all the artifacts that need to be isolated before implementation. The third option targets for existing Web application vendors (Guo, Sun, Huang, Wang, & Gao, 2007; Sun, Zhang, Guo, Sun, & Su, 2008; Cai, Zhang, Zhou, Cai, & Mao, 2009). In this approach, an existing Web application will be transformed to be multi-tenancy enabled. At the same time, application server runtime and database server runtime will be instrumented to route the requests for different tenants. This option could relieve the burden of application vendors who have existing Web applications with massive customers using them. These application vendors could use the SaaS multi-tenancy enabling tool and runtime to try their application in SaaS market quickly. In this way, they expect to serve more customers while still