SaaS Multi-Tenancy Isolation Testing-Challenges and Issues

Avneesh Vashistha, Pervez Ahmed

Abstract -Cloud computing is broadly categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Multi-tenancy becomes an important feature of SaaS, designing and building of multi-tenancy aware applications introduces several new challenges, central one is tenant. A service provider can support multiple tenants at the same time, while from a customer point of view; the tenants are isolated and customized for their specific needs. In this paper we present multi-tenant isolation testing challenges and issues.

Keywords – Multi-tenancy, IaaS, PaaS, SaaS, Isolation.

I. INTRODUCTION

Software as a Service (SaaS) is defined as "the capability provided to the consumer to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings" [8] .

In [2] Four SaaS maturity levels are being introduced, relating to how the SaaS application is delivered to many customers (tenants). At the first level, each customer has its own customized version of the SaaS application, and run its own instance of the application on the host's servers. This level corresponds to the traditional ASP [3] model. While in the second level, the vendor hosts a separate instance of the application for each customer (tenant). Despite being identical to one another at the code level, each instance remains wholly isolated from all others. At the third level of maturity, the vendor runs a single instance that serves each customer, with configurable metadata providing a unique user experience and feature for each one. At the fourth and final level of maturity, the vendor hosts multiple customer on a load-balanced farm of identical instances, with each tenant's data kept separate, and with configurable metadata providing a unique user experience and feature set for each customer.

II. RELATED WORK

Ralph Mietzner *et al*. [6] describe how the chosen patterns influence the customizability, multi-tenant awareness and scalability of the application. Using these patterns researchers describe how individual services in a multi-tenant aware application cannot be multi-tenant aware while maintaining the overall multi-tenant awareness of the application. Chang Jie Guo *et al*. [5] explore the requirements and challenges of the native multi-tenancy pattern which have the potential of serving a large volume of clients simultaneously. They provide a framework with a set of multi-tenancy common services to help people design and implement a high quality native multi-tenant application more efficiently. Harris and Ahmed [7] highlighted that how SaaS with its multi-tenancy approach along with efficient utilization of SOA leveraging the enterprises. In this paper they proposed an open multi-tenant architectural blueprint based on a real world scenario.

III. SAAS MULTI-TENANCY CHALLENGES AND ISSUES

Multi-tenancy is one of the key characteristic of the SaaS application. By leveraging Multi-tenancy, SaaS providers can significantly ease operations and reduce delivery cost for a big number of tenants. Multi-tenant applications provided with a single application, shared by multiple customers. In it a configuration file is being created and deployed every time a customer places request for services on multi-tenant application [4].

Both, Functional and non-functional testing can be applied on a SaaS application. Functional testing includes exploratory testing, end to end business workflow testing, automated regression testing, data integration and data migration testing, and checklist testing. On the other hand non-functional testing supports security testing (e.g., application, network, user access and role testing, data/security integrity testing etc.) and performance testing (e.g., scalability, volume, availability, reliability testing etc.). But moving beyond functional and non-functional requirements, emphasis also needs to be laid on testing the operational aspects like compatibility testing, live testing and SaaS attribute testing of the SaaS application. In this aspect what is the need of SaaS Attribute like Multi-tenancy isolation concept?

Isolation should be carefully considered in almost all parts of architecture design, from both non-functional and functional level, such as security, performance, availability, administration etc. Secondly, we should support tenants customize their own service in runtime without impacting others. Traditionally, customization would involve code modifications and application re-deployment. However, such a customization pattern is impractical in a multi-tenancy-enabled service environment. As all the tenants share the same application instance, once the customization is done for a particular tenant, the services for all tenants will be affected, and possibly interrupted during the update. As the number of tenants increases, the interruptions become more frequent and lead to very serious service availability issues.

Manuscript Received on November, 2012.

Avneesh Vashistha, Ph.D-CSE, Research Scholar, Sharda University, Greater Noida, India.

Dr. Pervez Ahmed, Professor , Department of CSE ,Sharda University, Greater Noida , India.

Therefore, the customization for one tenant without impacting other clients during the runtime should be a key requirement of a native multi-tenant application [5].

3.1 Technical Isolation Challenges Faced by Solution Developers Include [1]-

a) Access Control- how can application resources, e.g., virtual portals; database tables; workflows; web services- be shared between tenants so that only users belonging to a tenant can access the instances belonging to that tenant?
b) Customizability –
    - Database- how do we customize a shared database schema for one tenant without affecting others?
    - User Interface- how can we enable customization of the software look through configuration only?
    - Business Logic- how do we allow the business logic to be customized for each tenant without code changes?
    - Workflows – how do we let tenants customize the assignment of human tasks and other conditional tasks in shared workflows?
    - Tenant Provisioning- how do we automate the provisioning of a new tenant?
    - Usage based metering- how do we record usage of a service so that each tenant can be changed only for the usage of a service.

3.2 Technical Isolation Challenges Faced by Service Providers Include-

a) Database sharing, customization, backup, and restore of tenant-specific data- how can service providers choose between different database partitioning schemes based on performance, management, and scalability criteria?
b) Rapid enablement of multi-tenancy for existing web services- how can single- tenant web services implementations be enabled for multi-tenancy with little or no code changes?
c) Managing connectivity between a large number of third-party service providers and departmental service consumers in a large enterprise- different department (such as credit and mortgage loan departments) in the banking-service provider enterprise may use different credit-check service providers. How can the central IT department monitor, authorize, and meter the usage of the multiple credit-check services by different departments in the enterprise.
d) Scalability, improved hardware usage, and tenant-specific quality of service (QoS)- how can service providers improve the usage of hardware that's shared between different tenants and provide scalability?

IV. MAJOR CONCERNING MULTI-TENANCY ISOLATION TESTING ISSUES

There are three different methods for achieving multi-tenancy: using a database, using virtualization or through physical separation. Using either of the said category multi-tenancy isolation testing can be achieved on the following issues such as security, resource, performance, availability/scalability, administration, customization, data, execution and versioning.

V. CONCLUSION AND FUTURE WORK

This study intends to provide recommendations for both providers and customers- in terms of what to consider and how to manage multi-tenancy isolation testing activities for SaaS in the cloud environment. In our future work we are going to develop a model, for each isolation issue as discussed in section 4.

REFERENCES

1. http;//www.ibm.com/developerworks/library/ws-middleware/
2. F. Chong and G. Carraro, "Architecture Strategies for catching the Long Tail", Microsoft Corporation, http://blogs.msdn.com/gianpaolo,April 2006.
3. L. Tao. Shifting paradigms with the application service provider model. Computer, 34(10):32–39, 2001.
4. S.Merkel, "Parallels Software as a Service(SaaS)," p.2.
5. Guo C.J. Sun W., Huang Y., Wang Z.H., and Gao B., " A Framework for Native Multi-Tenancy Application Development and Management" proceeding of 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 2007, pp. 1-8.
6. Mietzner R., Unger T., Titze R., Leymann F., " Combining Different Multi-Tenancy Patterns in Service-Oriented Applications", proceedings of IEEE International Enterprises Distributed Object Computing Conference,2009, pp. 131-140. SP800-145.pdf.
7. Harris I.S., Ahmed Z., "An Open Multi-Tenant Architecture to Leverage SMEs", published in European Journal of Scientific Research, 2001, pp. 601-610.
8. http://csrc.nist.gov/publications/nistpubs/800-145/