Multi-Tenant, Secure, Load Disseminated SaaS Architecture

Zeeshan Pervez, Sungyoung Lee

Department of Computer Engineering, Kyung Hee University, South Korea

{zeeshan, sylee}@oslab.khu.ac.kr

Abstract – The availability of high speed internet has diversified the way we used to intermingle with each other. The emergence of social networks and interactive web applications has left a dent in existing software and service delivery models. Software vendors now not only focus on functionality but also have to cater delivery model of their software. On-demand and ubiquitous accessibility has become an inimitable selling point for software vendors. In the last few years we have witnessed a term "Cloud computing" thronging blogs and search engines. Cloud computing is on-demand service delivery; services ranging from Infrastructure, Platform and Software. With the materialization of Amazon Cloud Computing Service we have seen exponential increase in interest of business as well as research community in Cloud computing Orchestra. Now technological oracles are offering their software in Cloud as Software as a Service (SaaS). Every service provider is laying on line to gain competitive advantage over each other, there is need to delineate development guideline for SaaS. Without any doubt security is one of the main concerns for Cloud computing environment but unfortunately in the mist of security issues general recommendations for efficient Cloud has faded away. In this paper we have proposed a Cloud computing architecture focusing SaaS, which provides general specifications for SaaS design and for services implemented in it.

1. INTRODUCTION:

Over the time we have seen some dramatic changes in software delivery model; from stand alone applications to client server architecture, from distributed to service oriented architecture (SOA). All of these transformations were intended to improve ease of use and to make business process execution efficient. New software delivery model emerges due to the fact that either the earlier delivery model were not supporting the business need or technological advanced has broken some barriers which were considered to be inevitable in the previous ones. Exponential increase in processing power of enterprise servers, adoption of virtualization, service oriented architecture and availability of high band width to the masses has introduced new type of software delivery model known as Software as a Service (*SaaS*). SaaS is a software delivery model, which provides customers access to business functionality remotely (usually over the internet) as a service [1]. SaaS addresses issues like service abstraction, reliability, data accessibility and interoperability.

Leading companies in information technology industry are gradually moving their applications and related data over the internet and delivering them through SaaS [2]. Google has used SaaS platform to offer web applications for communication and collaboration [3], gradually replacing recourse exhaustive desktop applications. Similarly Microsoft is offering their development and database services though SaaS platform codename Microsoft Azure [4]. SaaS is preached by the companies like SalesForce.com, 3Tera, Microsoft, Zoho and Amazon, as a result of which business specific services can be consumed in ubiquitous environment.

2. MULTI-TENANT ARCHITECTURE:

SaaS not only has altered the software delivery model, but it also had amended the way in which software is developed. SaaS introduces the concept of Multi-Tenant, in which same software is used by multiple users or even by the different enterprises at the same quantum of time.

SaaS can be categorized into four levels of maturity [5]. First level services are *Ad Hoc/Custom* which dictates that separate instance of a service is deployed for each tenant. At second level services are *Configurable*; it is similar to the previous level, except that each service is configured according to the tenant business need; apart from that the core functionality provided by every service remains the same. Third level is *Configurable, Multi-Tenant-Efficient*; the previous two levels did not support multi tenant, but at this level services are designed to be accessed by multiple service consumer at a same time, without compromising the data of each tenant. At the fourth level services are *Scalable, Configurable, Multi-Tenant-Efficient.* This level is same as previous level, but additionally support multi tenant and services are configurable with the business need. This level focuses on the quality of service, and Service Level Agreement (SLA) which is signed between service provider and service consumer.

All of the above mentioned levels categories fall under the category of SaaS. The difference is how these services are provisioned to be consumed by the service consumer. Undoubtedly last level is the more preferable from service provider because of its maintainability and scalability point of view.

3. SAAS AND PROPOSED CLOUD COMPUTING ARCHITECTURES:

Cloud computing is broader term which included Infrastructure as a Service, Platform as a Service and Software as a Service [6]. It has become a unique selling point for products which are in their development phases and even to the new versions of existing products. Recently Microsoft has release their office suite in cloud computing environment [7]. Service consumer can consumer MS Office, thus leveraging their applications with SaaS. Service providers are talking about moving data center and commutation intensive applications to cloud.

Search engine around the world receive enormous number requests about cloud computing. Figure 1 shows the result of Goolge Trend showing the popularity of Cloud Computing in year 2009.

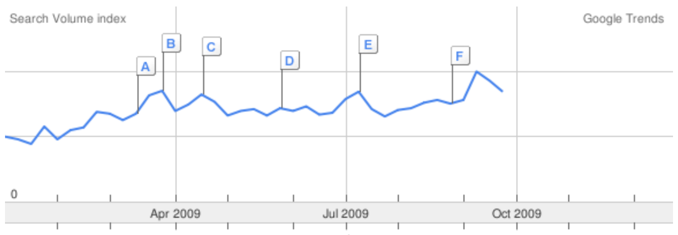


Figure 1 Google Trend for Cloud Computing

Whenever new technology or technique is introduced it not only works as a remedy for the existing problems; but it also brings in new threats and challenges. Same is the case with SaaS. CIO and CTO are rolling their sleeves up to adopt Cloud computing, and are offering their software over the wire as a service. On the other side service consumers are adopting Cloud computing in their application, to empower their applications with enterprise services at very cheap rates.

A large number of research material has been published discussing Cloud computing and it potential usage in current business scenario. Ample effort has been made by Cloud computing evangelists to clearly draw a line between IaaS, PaaS and SaaS. [1], [8] and [9] presents their Cloud computing model which address different issues ranging from over all orchestra of Cloud computing, to security and performance issues in it. Most of the research is focused on what and what should not be delivered as s service, and talk about mainly IaaS and PaaS. Unfortunately none of them discusses how different components in Cloud computing architecture interact with other form SaaS point of view.

SaaS could be one of the stepping stone towards the adoption of Cloud computing. There is a need to propose a Cloud computing architecture focusing on SaaS; which identifies what are the necessary components and how would they interact. As a result introducing robustness, reliability, performance to Cloud computing and bring peace of mind to service consumer from security of view. There is lack of end to end solution for SaaS based applications which provides guideline for service consumers and service providers; how has step forward to embrace Cloud computing for their applications.

4. MULTI-TENANT, SECURE, LOAD DISSEMINATED SAAS ARCHITECTURE

Here is this paper we present a cloud computing architecture especially focusing on Software as a Service (SaaS). Our proposed architecture is developed considering security and load balancing requirement in cloud computing environment. Apart from that one of the main factor which was considered during the designing phase of this proposed architecture was Multi-Tenant nature of Cloud computing. Multi-tenant is essence of Cloud computing, but injecting multi-tenant support in software poses some challenges not only to service provider but to service developer as well. With the help of proposed architecture; which we called Multi-Tenant, Secure, Load Disseminated SaaS Architecture (MSLD), we have made software development and provisioning tranquil for both stake holders. Figure 2 shows the proposed architecture.

Core focus of MSLD is to make realized service as light as possible in terms of service delivery. With MSLD, realized services in cloud computing environment no longer have to deal with security aspect in multi-tenant delivery model. Apart from security concerns MSLD take off unnecessary execution from the realized services and disseminate them to the appropriate services which are distinctively designed for this exclusive purpose.

MSLD is divided into five following services depending upon their functionality in Cloud computing orchestra.

- Responder Service
- Routing Service
- Security Service
- Logging Service
- Service Realization

The purpose of these services is to make the MSLD architecture loosely coupled in order to achieve seamless integration and service up gradation. Apart coupling and cohesion, these services are also aligned with the general requirement of Cloud computing, that is on-demand resource provisioning and service virtualization.

A - Responder Service:

It works as an entry point in MSLD. The core functionality of this service is to entertain request originating from diverse type of service consumers. Due to nature of heterogeneous environment of cloud computing this service is designed to handle different type of request, ranging from Personal Digital Assistance (*PDA*) to personal computers to services hosted in Cloud by other enterprise (*public and private cloud*).

Consider a case in which MSLD hosts NASDAQ Core Service (NCS) [12]; these types of services provide functionality to diverse consumers. As NCS will be offered as SaaS; there is a need to categorize (mobile, desktop and enterprise request) request generating from different service consumer mentioned earlier. All of these service consumers have their benefits and limitations, because functionality offered on PDA application will be limited as compared to personal computer, on the other side the requirement of cloud is far greater than of personal computer.

To handle diverse type of request we have introduced Response Generation Component, which generates response according to the requester capabilities. By adding this component to Responder Service we have added an extra layer for abstraction for service developer. Now there is no need to write code for handling different types of service consumers at service level.

By virtue of Responder Service, service developer can focus on business process instead of wasting their efforts on communication layer, and writing separate components for each category of service consumer. One of the benefits of this type of categorization; service provider can apply different type of Service Level Agreements (SLA) on different categories.

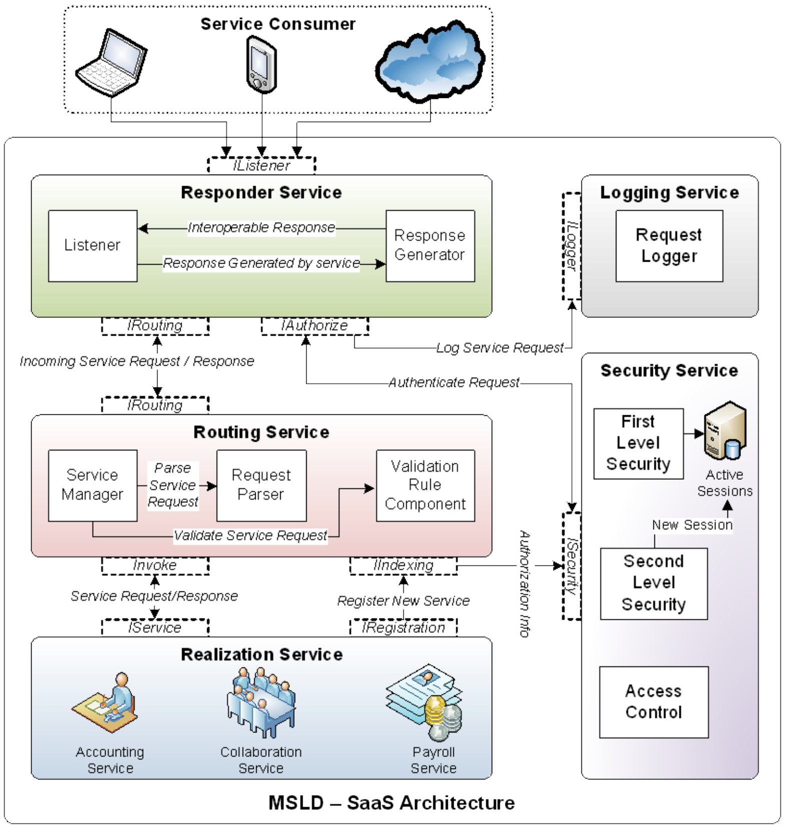


Figure 2 - Multi-Tenant Secure Load Disseminated Software as a Service Architecture

B - Routing Service:

This is one of the main component is MSLD, it indexes every service hosted by MSLD, besides this it validates the request in order to reduce the execution over head on the service next to it. Routing Service is composed of three components Service Manager, Request Parser and Validation Rule Component. Each of these components inter communicate with each other to rout incoming request to realized services. Routing Service takes on the responsibility of request validation as well.

Service Manage (SM) works like an indexer for every service and it keeps record of functionality offered by them. Functionality depends on the service implementation; service targeted to business process mostly exposes more than one interfaces. Request Parser (RP) parses the incoming request to check whether it conforms to the defined XML schema. XML schema is maintained for every realized service which contains at least one subschema for the functionality provided by the realized service. This type of validation helps in identifying the missing fields in incoming request earlier then it is actually executed. Validation Rule Component (VRC) is a repository which is used to validate incoming request but at the higher level of degree as compared to RP. VRC describes business rules for the service like valid range of values and valid data types. Table 1 shows the example of service validation file.

Considering the same example discussed in previous section in which MSLD hosts NASDAQ Core Service, which itself composes of multiple services like Audio Webcasting, Dynamic Annual Reports, Insurance Benchmarking, Press Release Distribution and Board Management Solutions. All of these services could be replicated in the virtual environment depending on their demand curve. So there is need to keep track of each service and to identify which request should be transferred to which instance. There are some cases in which variant services offer different request with same name, as they are developed independently. Suppose that Dynamic

Annual Reports and Insure Benchmarking offer different functionality with same name like *GetStockStatistics*. To manage this duplication SM, RP and VRC interact with each other and decide request belongs to which service. In this decision making process request is parsed to check it compliance with which service.

Table 1-Service Schema File

```
<ServiceName>

  <GetStockStatistics>
    <Input_Parameters>
      <Parameter_1 Name =’StockId’
        Value=’DT_0019’ Type=’NString’/>
      <Parameter_2 Name =’Date’ Value=’5/7/2009’
        Type=’Date’/>
      .
      .
      .
    </Input_Parameters>
  </GetStockStatistics>

  <Functionality_Name>
    <Input_Parameters>
      <Parameter_1 Name =’’ Value=’’ Type=’’/>
      <Parameter_2 Name =’’ Value=’’ Type=’’/>
      .
      .
      .
    </Input_Parameters>
  </Functionality_Name>

</ServiceName>
```

C - Security Service:

Security has been the concern of information technology industry. Every new technology has to brawl with it to make it adaptive within the society. Same is the case with Cloud computing. When switching to the Cloud computing platform; irrespective of it nature (*Infrastructure as a Service, Platform as a Service or even Software as a Service*), service consumers always ask about their data security and confidentiality.

In MSLD design Security Service is dedicated for security purpose which controls authentication and authorization process. Every incoming request has to pass through the Security Service. Other services like Responder service and Routing Service depends on its response. In short Security Service validates that whether coming request is from legitimate user. In addition to this it also confirm that whether the requester posses the rights to consume the service.

MSLD introduces two level of security Level-I and Level-II. First one deals with sessions which have been active for a while; and later the newly generated sessions. The purpose of introducing two level of security is to hike up system performance and to increase user experience and certainly to establish more reliable security measures.

Responder Service on receiving request form service consumer delegates to Security Service, which checks it for Level-I security. If it qualifies it then begins the authorization process to check whether the service consumer has privileges to access requested service. During this process if any of the qualifying tests is not passed by the request it is send back to Responder Service which then propagate error message to the service consumer.

The Level-II security deals with newly created session, when user login to the system first time to create a valid session it is routed to Level-II security which authenticate the requester from valid customer repository, and then grants the permission to consume the service.

MSLD does not impose any restriction on how authentication or authorization should be implemented. For authentication any Single Sign On (*SSO*) protocol can be implemented. SSO is preferred, as software is being offered as a service, so multiple requests could be send to the realized service in order to complete single business transaction, in this scenario SSO is an ideal authentication process. For the authorization process MSLD does not restrict service implementer to any specific authorization model. Any authorization model can be implemented like Mandatory Access Control, Discretionary Access Control.

Each time service in registered to Routing Service it also provides the authorization rules to Security Service. Table 2 shows the example of authorization information. Whenever new service consumer is registered to a service, it is recorded in Security Service. By isolating this type of functionality form realized services we can reduce execution load on them and hence can improve quality of service.

Table 2 Service Authorization Information

```
<ServiceName>
  <GetStockStats>
    <Subscribers>
      <Name = ‘Company A’ Access_Rights=’R’>
      <Name = ‘Enterprise XYZ’
        Access_Rights=’RWE’>
      .
      .
      .
    </Subscribers>
  </GetStockStats>

  <Functionality_Name>
    <Subscribers>
      <Name = ‘’ Access_Rights=’’>
      <Name = ‘’ Access_Rights=’’>
      .
      .
      .
    </Subscribers>
  </Functionality_Name>
</ServiceName>
```

D - Logging Component:

The logging component is multifaceted component is MSLD. Firstly it serves as an auditing component in case of service failure. This helps in identify the source and reason of service failure which ultimately helps in meeting SLA and to improve quality of service. Secondly the solution offered in Cloud follows the business model of pay-as-you-go; you pay only what you have used. In this situation logging component proved to be utilitarian, it logs every request entertained by the architecture, which in the end facilitates in charging the client for what they have used instead of subscription based.

This is one case in which service consumer is charged on per request basis; additionally one could come up with a new pricing model in which data generated by the service (*response*) could be used to charge the service consumer. It depends on the service provider how it uses this factor, more data more cost or more data less cost. By introducing such pricing model service provider can gain competitive as well strategic advantage on traditional software vendor. This could be one of the driving forces to encourage service consumer to migrate their application or data to cloud computing environment.

E - Service Realization:

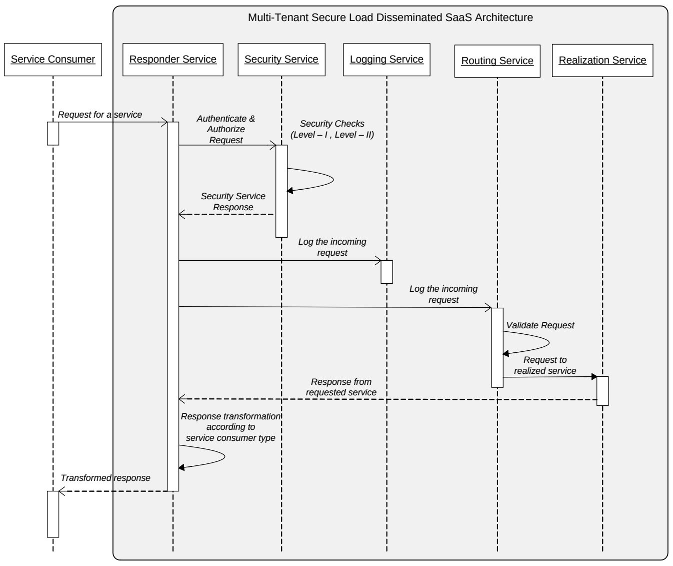
Service Realization hosts the implemented services which will be exposed to the service consumer. This service is implemented keeping in view the Multi-tenant nature of Cloud computing (SaaS). As described earlier MSLD unties functionalities which could cause hindrance in adoption of Multi-tenant from the actual service implementation. We have shown how security and load balancing has been disseminated around the architecture instead of implementing it in the service itself. Any type of service can be hosted in Cloud ranging from simple tax calculation service to complex enterprise services like procurement, accounts, and billing service. It exposes an interface to the Routing Service, on receiving the incoming request; it routs it to the appropriate service.

At the very initial level MSLD delegates the functionality of handling the incoming request to Responder Service. By separating this logic from actual service we have leverage the service developer to focus only on service implementation and not on making service interoperable to diverse service consumers.

In multi-tenant environment access control is one of the concerns, service developers' deals with; by stripping off the access control logic from service implementation we have reduced the complexity of access control from actual service. Now service developers only describes the access control rules in the security component during registration process; and in the actual execution security component automatically validates them on behalf of service realization. Untying access control logic from service implementation also helps in registering new tenant to the service, now there is no need to code extra access control checks each time new tenant is registered.

5. REQUEST PROPAGATION IN MSLD

MSLD architecture is based in SOA; each service exposes an interface to communicate with other service. Figure 3 shows how services in MSLD architecture interact with each other to provide services to consumer whilst addressing security concerns of service provider and service developer.



Multi-Tenant Secure Load Disseminated SaaS Architecture
Service Consumer
Responder Service
Security Service
Logging Service
Routing Service
Realization Service
Request for a service
Authenticate & Authorize Request
Security Checks (Level – I , Level – II)
Security Service Response
Log the incoming request
Log the incoming request
Validate Request
Request to realized service
Response from requested service
Response transformation according to service consumer type
Transformed response

Figure 3 Interaction between difference services of MSLD

Service consumer requests for a service by interacting with responder service. On receiving the request form service cons umer, it checks whether the request is generated from the legit imate customer and has valid session. To validate the request Responder Service routs the request to Security Service, whic h confirms the requester authenticity. First it will check whet her user has the valid Session Id and has the privileges to exec ute the requested service. If session id is not valid or does not exists in the local cache then it will ask its credentials, on succ essfully proving it identity newly created Session Id is stored i n local cache for upcoming request to boost up the processing time.

Once the request passes the security it is being logged by th e Logging Service with all of its details like service consumer info, requested service, time of request and request parameters. Then it is being forward to Routing Service, to avoid unneces sary service execution on invalid request, incoming request is validated against the request schema file which list all of the n ecessary parameters along with their data types and valid rage of values (*business rule*).

On successfully conforming to the defined rules request is passed on the Realization Service which checks the request an d pass it to the appropriate service. Once the execution proces s is completed response is send back to the responder service. On receiving the response from the realized service it check th e type of service consumer; to transform the response accordi ng. This ensures the response is generated according to the ca pabilities of requester. This feature MSLD ensure that single i nstance of a service is used to serve different type of service c onsumer. Besides this it also facilitate the service developer to focus on service implementation instead of trying to achieve i nteroperability.

6. CONCLUSION

MSLD is a SaaS targeted Cloud computing architecture which focuses on over all service interaction form service consumer and service provider point of view. MSLD address the issues of security, performance and load management in SaaS. MSLD sets general guide lines for service interaction and does not dictates any type of communication protocol and security technique, it totally depend on the service provider how he provision hosted services. Moreover it shows how request from ubiquitous environment could be efficiently handle by Cloud computing environment without the need to alter service implementation. MSLD also highlights the fact that by extracting authorization, authentication and validation logic form the realized service and distributing them to the appropriate layer could greatly improve the performance which ultimately helps in increasing user experience and meeting SLA.

7. ACKNOWLEDGEMENT

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2009-(C1090-0902-0002)). Also, it was supported by the IT R&D program of MKE/KEIT, [10032105, Development of Realistic Multiverse Game Engine Technology].

8. REFERENCES

[1] Sun, W., Zhang, K., Chen, S., Zhang, X., and Liang, H. 2007. *Software as a Service: An Integration Perspective.* In Proceedings of the 5th international Conference on Service-Oriented Computing (Vienna, Austria, September 17 - 20, 2007).

[2] Zhang, L. and Zhou, Q. 2009. *CCOA: Cloud Computing Open Architecture*. In Proceedings of the 2009 IEEE international Conference on Web Services - Volume 00 (July 06 - 10, 2009).

[3] Google Web Applications for Communication and Collaborations. http://www.google.com/apps

[4] Microsoft Windows Azure Platform. http://www.microsoft.com/azure/default.mspx

[5] http://msdn.microsoft.com/en-us/library/aa479069.aspx#docume_topic10

[6] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. 2009. *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility.* Future Generation Computer Systems. 25, 6 (Jun. 2009).

[7] http://www.microsoft.com/presspass/features/2009/sep09/09-17officewebapps.mspx

[8] Hudli, A. V., Shivaradhya, B., and Hudli, R. V. 2009. *Level-4 SaaS applications for healthcare industry*. In Proceedings of the 2nd Bangalore Annual Compute Conference on 2nd Bangalore Annual Compute Conference (Bangalore, India, January 09 - 10, 2009). COMPUTE '09

[9] Karabulut, Y. and Nassi, I. 2009. Secure Enterprise Services Consumption for SaaS Technology Platforms. In Proceedings of the 2009 IEEE international Conference on Data Engineering (March 29 - April 02, 2009). ICDE. IEEE Computer Society, Washington, DC, 1749-1756.

[10] F. Chong, G. Carraro, R. Wolter. Multi-Tenant Data Architecture. June 2006.

[11] http://msdn.microsoft.com/en-us/library/aa479086.aspx

[12] NASDAQ Core Services. http://nasdaqomx.com/whatwedo/servicesforcompanies/usmarket/coreservices/