A USER INTERFACE PROTOTYPE OF TEST SUPPORT AS-A-SERVICE

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Neha Kale

In Partial Fulfillment for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

August 2012

Fargo, North Dakota

North Dakota State University
Graduate School

Title

A User Interface Prototype of Test –Support-as-a-Service

By

Neha Kale

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr.Tariq King
Chair

Dr.Kendall Nygard

Dr. Jun Kong

Dr.Limin Zhang

Approved:

08/06/2012	Dr.Kendall Nygard
Date	Department Chair

ABSTRACT

Cloud computing provides software as a service over the internet. With the increasing popularity of cloud applications, the development of error-free cloud-based applications that function reliably is necessary. Software development under the cloud computing model brings the advantage that new applications can be rapidly constructed by tailoring existing services. However, such a development model reduces the testability of the application. To address this problem the approach of Test Support as-a-Service (TSaaS) is proposed.

This paper presents a user interface prototype for the TSaaS approach to investigate the feasibility of the proposed TSaaS approach. The paper presents the user interface design for the TSaaS approach and compares the approach with the other existing approaches. In this way, with the help of proposed user interface, one can now test the integration of cloud-based services effectively with the help of third service TSaaS.

ACKNOWLEDGEMENTS

This research project could not have been possible without the support of many people. First of all, I would like to express my deepest thanks to Dr.Tariq King for guiding me in my research. The 'open door policy' and discussing the issues helped me achieve this task. I would like to express my very great appreciation to Dr. Kendall Nygard for his advice and assistance in keeping my progress on schedule. At the every stage of development in my study he lend me a helping hand without his support this journey could not be possible. I also would like to thank my committee members Dr. Jun Kong, Dr. Limin Zhang and Dr. Kendall Nygard for their time and suggestions. I would also like to extend my thanks to all the members of Software Testing Research Group (STRG). I specially would like to thank Annaji Sharma for his guidance, support, and suggestions during this work as well as encouraging me to think and work independently. His willingness to give his time so generously has been very much appreciated.

My special thanks to my husband Dr. Rajan Bodkhe for his constant support and motivation. Furthermore, I wish to thank all my friends and family members for their support and encouragement throughout my work. Finally, I offer my regards and blessings to all of those who supported me in any respect during the completion of the research project.

DEDICATION

This paper is dedicated to

My beloved late grandmother Champavati Kale;

My Baby (is in my womb and is due on or around Dec 19 2012);

My loving husband Dr. Rajan Bodkhe

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

LIST OF ABBREVIATIONS

TSaaS……Test Support as-a-Service

UML…….....Unified Modeling Language

SaaS……...Software as-a-Service

PaaS…….....Platform as-a-Service

IaaS……Infrastructure as-a-Service

TaaS…….....Test as a Service

STITC…….....Software Testing In the Cloud

XML……Extensible Markup Language

API……..Application Programming Interface

GUI……..Graphical User Interface

OS…….Operating System

TSM……...Test Support Manager

TSR……...Test Support Repository

SOA…….Software Oriented Architecture

VHD……Virtual Hard Disk

LIST OF APPENDIX FIGURES

CHAPTER 1. INTRODUCTION

The popularity of software is quite evident from the millions of applications that are available worldwide. In recent years, with the increasing use of internet application, cloud services and applications are becoming popular. Cloud computing provides universal, on-demand access to shared computing resources via the internet. Cloud computing model brings the advantage that the new applications can be rapidly developed by using existing services.

However, the use of internet based services as a software component to build new services offers many challenges. One of the major challenges is to maintain the quality of the integrated service. With the use of efficient testing techniques, these failures can be avoided. Testing not only reduces errors but also helps in greater visibility of the software system and reduces the cost associated with software failures. However, the cloud-based environment services are remotely hosted, and there are many other challenges to testing the integration of two services. This paper highlights the challenges, the benefits associated with software testing of cloud- based services, and the solutions to resolve the problems.

1.1. Research Motivations

During the research study I learned that there are many testing challenges associated with testing cloud-based applications. In order to investigate the challenges associated with integration testing of cloud-based applications King et al [1] examined an application development and deployment scenario in the banking domain. They considered the small scenario where only two services were involved. Even with its narrow scope, the scenario revealed many of the integration testing challenges faced by the software vendor of the service. First, since the service is hosted remotely, controllability and visibility are limited. Second, the

vendor does not have access to the source code of the service, thereby limiting validation of its interactions with other components to black-box techniques. Lack of access to the source code of remotely-hosted services also complicates the process of diagnosing and debugging problems. Finally, the service that supposed to be available to the vendor for testing purposes is hosted in its production environment. To address all the above mentioned problems King et al [1] proposed an approach of Test Support as-a-Service (TSaaS) in the cloud. To further investigate the proposed approach in depth and to check the feasibility of the approach, designing and developing user interface for the proposed the TSaaS approach was an important challenging task in front of research team. I picked that challenge as my research idea. Along with design there was one more task of evaluation of the TSaaS approach to find out the positives and negatives of the approach. In this way, both of these challenges turned out to be a motivation points for my research.

1.2. Objectives

The research in this paper is focused toward three objectives: (1) the detailed description of a Test Support as-a-Service (TSaaS) application with the help of the architecture, detailed design and use case diagrams (2) implementation of a small user interface prototype that can be used to demonstrate the TSaaS approach's feasibility, and (3) evaluation of the TSaaS approach in context of other already existing approaches and cloud-based testing tools and discuss the positive and negative points of the TSaaS approach.

1.3. Outline of Paper

The rest of this paper is organized as follows: Chapter 2 provides the literature review information about cloud computing, testing in a cloud and also explores the current research in the area of integration testing of cloud-based applications. Chapter 3 defines the research problem. Chapter 4 describes the detailed description of Test Support as-a-Service (TSaaS). Chapter 5 explains the structure of the user interface prototype of TSaaS using architecture diagram, use case diagram and detailed design diagram to illustrate the behavior of the prototype. Chapter 6 discusses the evaluation matrix, results and discussion, the positives and negatives of the TSaaS approach. Finally, Chapter 7 concludes the paper and suggests future work.

CHAPTER 2. LITERATURE REVIEW

This chapter introduces the basics of cloud computing and software testing of cloud services. It discusses challenges and benefits offered by cloud testing, and covers classification of cloud testing. It also includes some ideas from other literatures inclined towards testing of cloud-based services.

2.1. Background

2.1.1. Cloud Computing

Cloud computing is an emerging paradigm to deploy and maintain software [2]. Cloud computing provides universal, on-demand access to shared computing resources as services over the Internet [3]. Services in cloud computing typically fall into three categories: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [3], [4]. SaaS describes software applications that are deployed in a cloud computing environment. SaaS, sometimes referred to as on-demand software, is a software delivery model in which software and its associated data are hosted centrally (typically in the cloud) and are typically accessed by users using a web browser over the Internet [5] and are primarily managed and controlled by a host provider, e.g., Google Docs, Microsoft Office 365 [6], [7]. The intended users for SaaS are end customer using SaaS applications. PaaS delivers programming languages and software development tools and operating environment to support the construction of SaaS. PaaS facilitate the deployment of applications without the cost and complexity of buying and managing the essential hardware and software and provides hosting capabilities, along with all of the facilities required to support the complete life cycle of building and delivering SaaS applications [8]. The provider allows the consumer to deploy, control, and manage cloud applications, and configure

the environment in which they are hosted, e.g., Microsoft Windows Azure [9]. The intended users for PaaS are developers. IaaS provides hardware, processing storage and networking capabilities for the purpose of hosting operating environments and applications in the cloud, e.g., Amazon EC2,Rackspace,GoGrid [10],[11],[12]. The intended users for IaaS are system administrators.

2.1.2. Testing in the Cloud

Testing requires lots of resources and the cloud is certainly powerful enough to handle it. Cloud Testing uses cloud infrastructure for software testing [13].Cloud offers certain benefits and challenges to testing. Some of the benefits offered by cloud testing are computational power, storage and virtualization. Some challenges of software testing in cloud are as follows:

1. Dependency on the Internet - Applications is not installed locally in controlled environments. This makes it harder for testers to replicate the user environment.
2. Security- Since information travels through the Internet, testers have to perform security testing to make sure there is no data leakage when data is sent over the Internet. Wikileaks is an example of threats that we have to prepare for and test before the application can be released to customers.
3. Testing all layers - testing the network connection, server performance, database, and software application adds multiple layers to testing. Testers have to test the communication between the layers, test the connection between the elements, and also plan for the risks.

2.1.3. Classification

The relationship between software testing and cloud computing can be divided into following categories:

1. Testing IN the Cloud - leveraging the resources provided by a cloud computing infrastructure to facilitate the concurrent execution of test cases in a virtualized environment
2. Testing OF the Cloud - validating applications that are hosted and deployed cloud
3. Testing TO the Cloud - moving the testing process and other assets to the cloud

2.2. Related Work

In this section we provide a literature review of works that are related to our research. Takayuki Banzai et al. [14] propose D-cloud, a software testing environment using virtual machines. Their model describes the use of cloud computing to test parallel and distributed systems after deployment as it is often more difficult to test parallel and distributed systems in the real world after deployment. D-Cloud system allows a user to easily set up and test distributed systems on the cloud with the use of virtual machines with fault injection facility. They used the XML to write the scenario of fault injection. D-Cloud system allows a user to easily set up and test a distributed system on the cloud and effectively reduces the cost and time of testing. Their definitions and testing criteria help to formalize problems associated with cloud testing and is complementary to our work.

George Candea et al. [15] propose a model for TaaS automated software testing as a cloud-based service. They describe three kinds of TaaS as a part of their research, a programmer's sidekick that enable developers to thoroughly and promptly test their code; a home

edition is on-demand testing service for consumers to verify the software they are about to install on their PC or mobile device; and a public certification service, akin to Underwriters Labs, that independently assesses the reliability, safety, and security of software. TaaS automatically tests software, without human involvement. Our work specifically addresses cloud computing, which provides a user friendly interface to access and reuse tests to minimize the human interaction.

Several works on testing cloud- based an application that includes testing tools and prototypes can also be founded in the literature [16], [17]. Liviu Ciortea et al. [16] propose Cloud9 which is an on-demand software testing service. It runs on compute clouds, like Amazon EC2, and measures its use of resources over a wide dynamic range, as compared with the testing task in hand. Michaela Greiler et al. [17] encourage the need for SOA online testing and though the series experiments with the cases study they showed that how online testing can detect many typical runtime reconfiguration faults, and how online testing provides additional value over offline testing.

CHAPTER 3. RESEARCH PROBLEM

This chapter gives an overview of the research problem under investigation to enable automated integration testing in cloud-based environments. In this paper, I will discuss the user interface prototype for the following research problem.

In Figure 3.1, host provider B is developing a service B that uses another already existing service A provided by provider A with the intention of extending service B's functionality and to speed up the application development process. This type of development model greatly reduces the testability of cloud services like service B as it has limited control over remotely hosted services like service A.

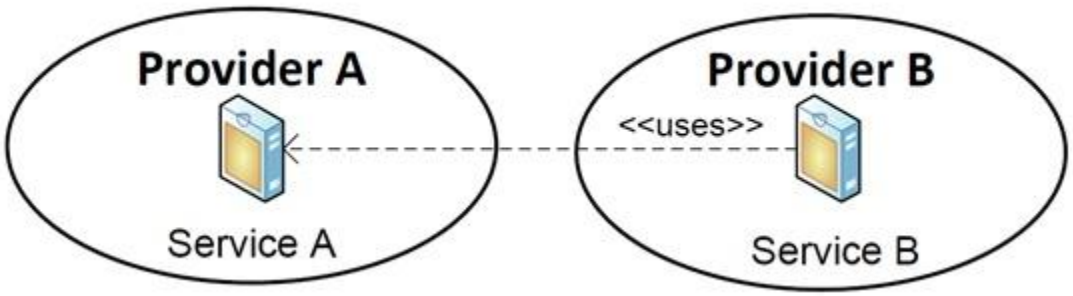

Figure 1. Integration of Cloud Services, Reproduced from King et al. [18]

Even with such a limited scope, integration of only two services, the following cloud integration testing challenges can be observed.

1. Difficult for provider B to setup tests for his service B since he does not have full control over service A.
2. Developing accurate test stub is hard since provider B has no knowledge of service A's implementation.

3. Provider B is only limited to the Black Box Testing of only parts of his service B implemented by service A.
4. Service A can change without provider B even knowing about change. Update to service A is important to know by provider B as service B needs to update and test accordingly.

To guide the investigation, following research questions were framed. The research questions are as follows:

1. How can pre-existing test artifacts (test driver, test stubs, and test data) and tools used to test service A can be reused to test service B.
2. How can built-in cloud computing technologies like virtualization be used to improve the process of automated integration testing of service B.

Solution to the above research problem is the Test Support as-a-Service (TSaaS) approach in the cloud proposed by King et al [1]. The TSaaS approach seeks to reuse test artifacts, tools and also use virtualization to support the configuration of test bed for services like service A. The aim of research is to develop the user interface .Net prototype for the TSaaS approach and to evaluate the approach against the already existing cloud testing tools in the market.

This paper focuses on the design of the user interface prototype for the TSaaS approach. Furthermore, to tackle the challenges associated with the feasibility of the proposed the TSaaS approach this research paper also concentrates on evaluating the TSaaS approach with the already existing cloud-based approaches and tools.

CHAPTER 4. TEST SUPPORT AS-A-SERVICE

4.1. Introduction to TSaaS

The research discussed in this paper is the part of ongoing research related to the Test Support as-a-Service (TSaaS) approach. This chapter explains the TSaaS approach as a solution to the research problem mentioned in the previous chapter 3. King et al [1] described a novel approach of TSaaS to enable integration testing of services in cloud computing environments. TSaaS provides cloud partners with access to automated test operations for remotely hosted cloud services [18]. The idea behind TSaaS is that, prior to the deployment of Service A (as shown in figure 3.1) in its production environment, Provider A would have validated the service to check for accurate functionality, interoperability, performance and other quality attributes. Due to the inherent drawbacks of manual testing, Provider A is likely to have employed automated software testing techniques during validation. If so, a pre-configured local test environment, test scripts to facilitate automatic test execution would be available on the infrastructure of Provider A. The TSaaS approach seeks to reuse elements of Service A's existing test automation to rapidly realize a set of test support services for Provider B. To avoid the security risks, TSaaS should be delivered via a protected interface that is only accessible to trusted collaborating providers.

TSaaS uses a separate copy of the service for testing. Using a separate copy for testing does not interrupt the regular operations of the service hosted in production environment. By allowing partners to observe and control private members of a copy of service A, TSaaS would enable integration testing. This ensures that the approach is applicable to services that have high availability requirements. The virtualization technique is incorporated into TSaaS to reduce the

overhead of handling concurrent execution requests from multiple partners. In an ideal world, the TSaaS implementation would need to be powerful enough to handle the concurrent execution of test support requests from multiple partners, each having a separate handle to a copy of the service under test and its test environment. This would lead to increased production overhead due to the need to set up and manage these additional computing resources. Therefore, to reduce the overhead of the proposed approach, the virtualization technique is incorporate into TSaaS.

In the next chapter, Chapter 5, I will explain the user interface for the TSaaS approach in detail with the help of UML diagrams.

CHAPTER 5. USER INTERFACE PROTOTYPE OF TSAAS

In order to evaluate the proposed research idea of the Test Support as-a-Service (TSaaS) approach, this chapter discusses the TSaaS user interface prototype in detail. This section presents the design and implementation of the TSaaS user interface prototype. The user interface prototype focuses on building a test infrastructure to help to improve the testability of cloud-based services. During integration testing, developers would access this interface to configure the test fixtures and test environment of the remotely-hosted service.

5.1. Application Description

To evaluate the approach by King et al. [18], a small scale service, Credit Reporting Service (CRS) was implemented, commonly referred to as Service A in this document. To provide a realistic context for the approach, I developed an application that provides user interface for the developers to perform the different operations proposed by the TSaaS approach with the help of Service A.

Following are the various operations that the TSaaS user interface provides to the developer to perform the integration testing of remotely hosted services.

Operation	Description
Configure Test Fixtures	User defines fixture operations
Configure Test Environment	User runs specified fixture operations
Runtime Virtualization	Facilitates create, start, stop, delete,take snapshot, and update VHD configuration
Server-Side Diagnostics	Execute tests and see the test results

Table 1: User Interface Prototype Test Support Operations

A User Interface Prototype of TSaaS consists of 5 main levels of use cases as shown in Figure 5.1. These use cases represent some core functionality of the system. Based on these core functionalities, the following description helps in demonstrating the main units of user interface design.

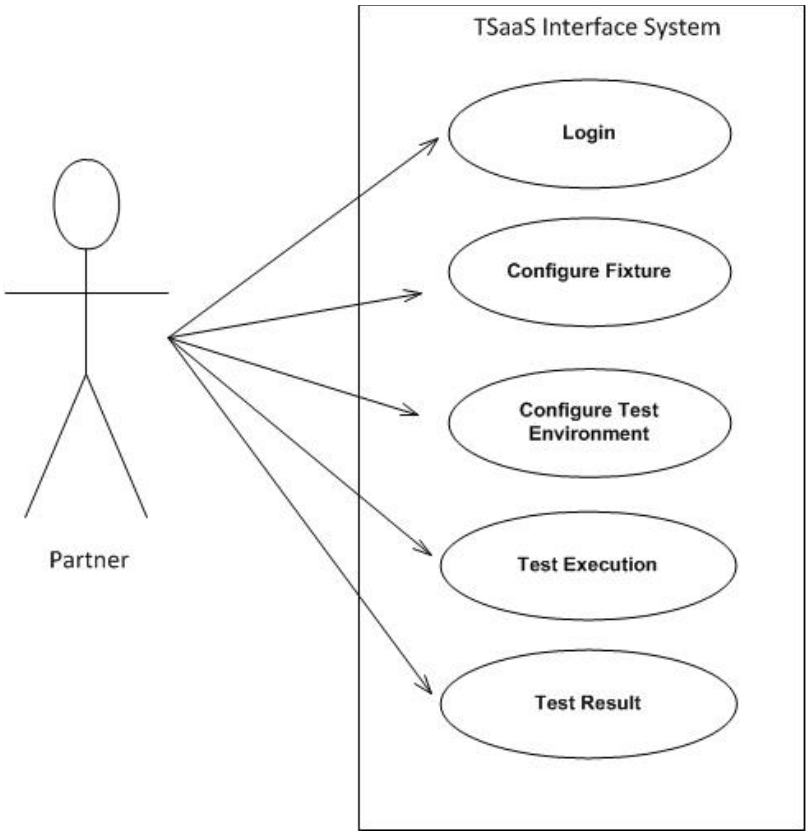

Figure 2. Use Case Diagram TSaaS

5.1.1. Login

According to the explanation given in the scenario in chapter 4, here onwards I will refer to Service A as a provider service and Service B as a partner service. In order to login to the TSaaS the partner must be authenticated. The Login functionality mentioned in the Figure 5.1 authenticates the partner by checking the credentials entered against the known set of valid users.

If the partner is valid, the partner will be given access to the TSaaS to perform the test operations.

5.1.2. Configure Test Environment

If the user is valid, the TSaaS prototype retrieves an image of the partner's virtual test environment from the Test Support Repository (TSR). From here the partner will be able to create the virtual test environment for the purpose of testing. This improves the availability of the cloud service being tested. With the help of virtualization, the partner can create a copy of the actual service and can perform different testing operations without disturbing the actual service in its working environment. The detailed process of configure virtual test server is explained in Figure 5.5.

5.1.3. Configure Test Fixture

This use case provides the partner the ability to configure the tests. Once the environment is configured, the partner can select the service to test and can configure the tests to run. The provider through TSaaS would provide a set of setup, teardown, input and assertions operations which can be used by the partner to configure test fixtures for developing integration tests. The partner through the user interface of TSaaS can design test fixtures for his test. The user interface provides a form that requires the partner to enter the TestID. The user interface provides a list of setup, teardown and control points (inputs and assertions) from which the partner can choose to configure a test. Figure 5.3 provides the screen shot for configure test fixture functionality.

5.1.4. Test Execution

This use case provides the partner the ability to execute the tests. From here the partner can test the application by selecting the tests to execute. The partner can either select one test at a time or he can select multiple tests. The partner can also select a “Run with Coverage” check box present on the user interface to run the tests with code coverage.

5.1.5. Test Result

This use case provides the partner an ability to see the test results. Here the partner can view the pass fail result of the test run along with the details of the error when the partner clicks to details link given under error section.

5.2. Architecture

This section describes the high level architecture of the Test Support as-a-Service (TSaaS) approach. As a part of the user interface prototype of TSaaS I mostly focused on developing presentation layer. From the services layer some services like virtualization, test support core and diagnostic were already developed. As a part of development of the user interface prototype of TSaaS I consumed those services. As a research part, I developed and consumed the Authentication service. I also interact with storage layer to retrieve user’s login credentials, virtual machine’s information and XML files.

Figure 5.2 provides the architectural design of the TSaaS prototype, including its major layers and their dependency relationships. The current version of the TSaaS prototype is organized according to the three-tier architectural style. The Presentation layer at the top of Figure 5.2 provides a set of ASP.NET pages that makes up the TSaaS GUI. During integration

testing, developers would access this interface to configure the test fixtures and test environment of the remotely-hosted service as shown in Figure 5.3.

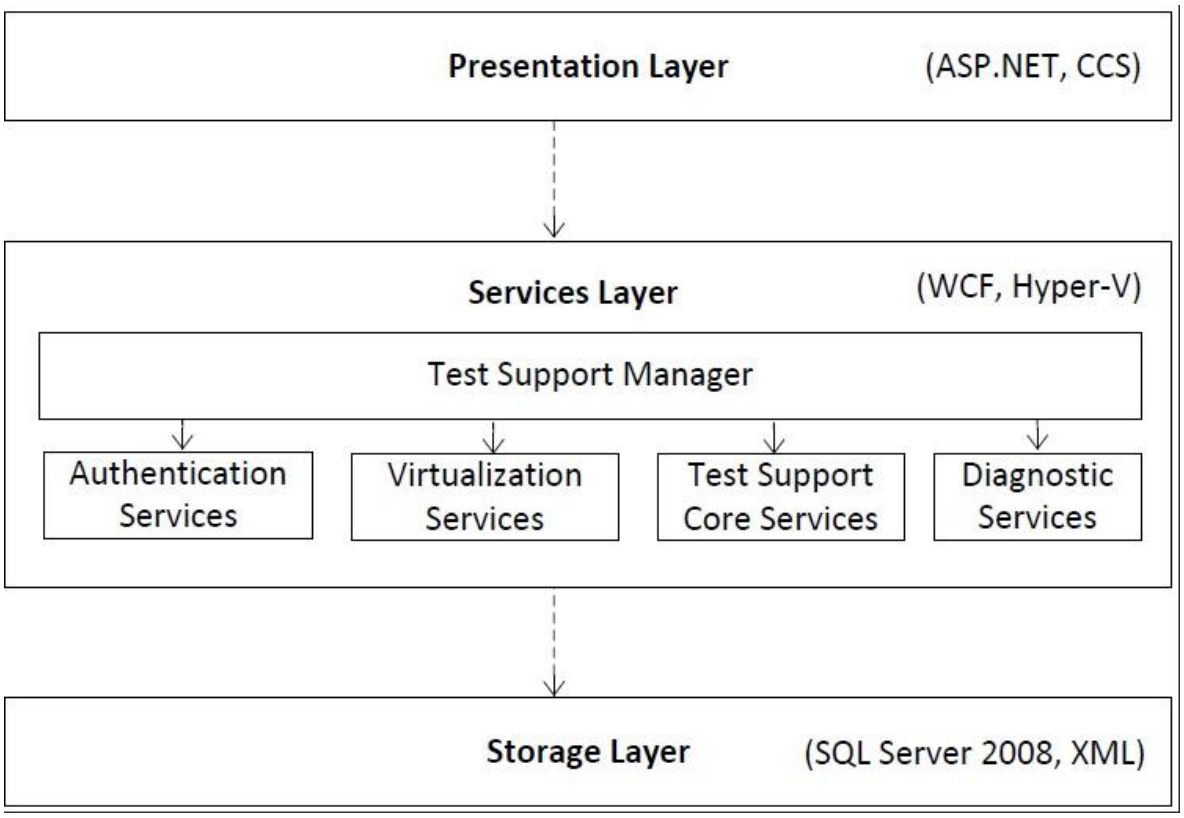

Figure 3. Architecture of the TSaaS Prototype

The Services layer in the middle is invoked by the presentation layer to realize user requests for test support. It includes a Test Support Manager (TSM) which coordinates services for: Authentication -- validates login credentials; Virtualization -- facilitates creating a virtual test environment of the remotely-hosted service; Test Support -- allows users to author, edit, and execute test fixture operations to support integration testing using a copy of the remotely-hosted service; and Diagnostics -- provides error reporting and server-side diagnostic features associated

with the remotely-hosted service. Services in this layer have been implemented using the Windows Communication Foundation and Microsoft Hyper-V technologies.

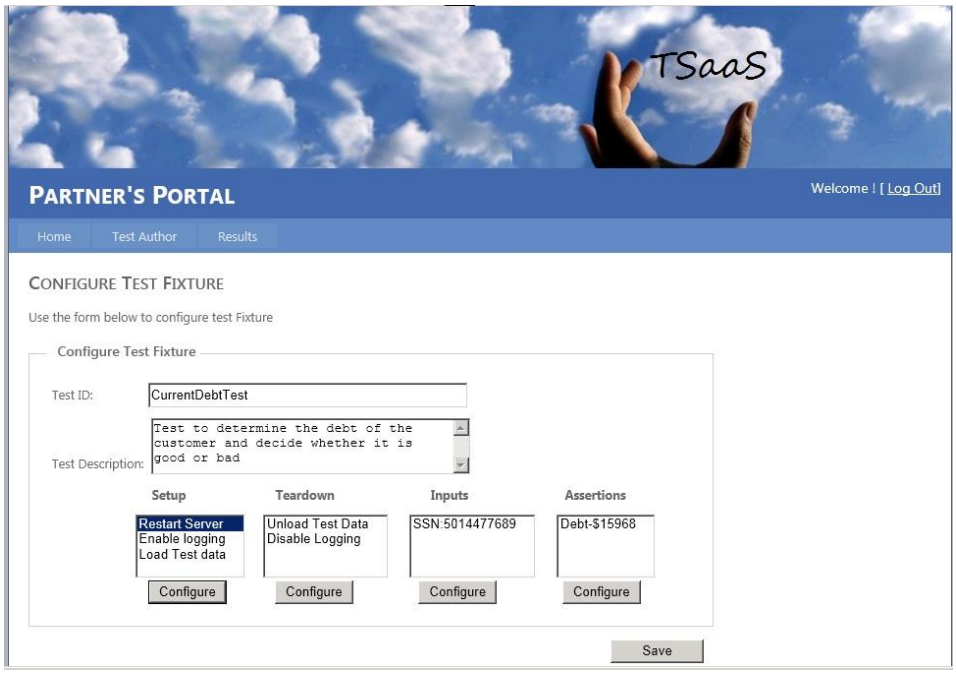

Figure 4. Test Fixture Configuration UI

The Storage layer at the bottom is used for all persistent data, including user accounts, virtual hard disks, test fixtures, and the results of verification points after running those fixtures within integration tests. Persistent user data is maintained in a Microsoft SQL Server 2008 R2 database, while test fixtures are defined using an XML-based test specification language.

5.3. Detailed Design

This section provides the detailed design for the TSaaS prototype. With the help of design-level model such as class and sequence diagram, this section focuses on giving a detailed description of the User Interface for TSaaS prototype behavior.

5.3.1. Static Model

The Figure 5.4 shows details about the different classes of TSaaS prototype their dependency interrelationships and the operations in each class. The TSaaS user interface includes 5 different classes: LoginPage, ConfigreFixturePage, ManageVMsPage, CreateNewServerPage and TestResultPage. Login class calls the TestSupportManager to access LoginController class from services layer that contains the Authentication operation for validation of user's login credentials. Similarly, CreateNewServerPage class calls TestSupportManager to invoke HyperVisorController class. Furthermore, ConfigreFixturePage and TestResultPage call TestSupportManager to invoke TestCoreController class.

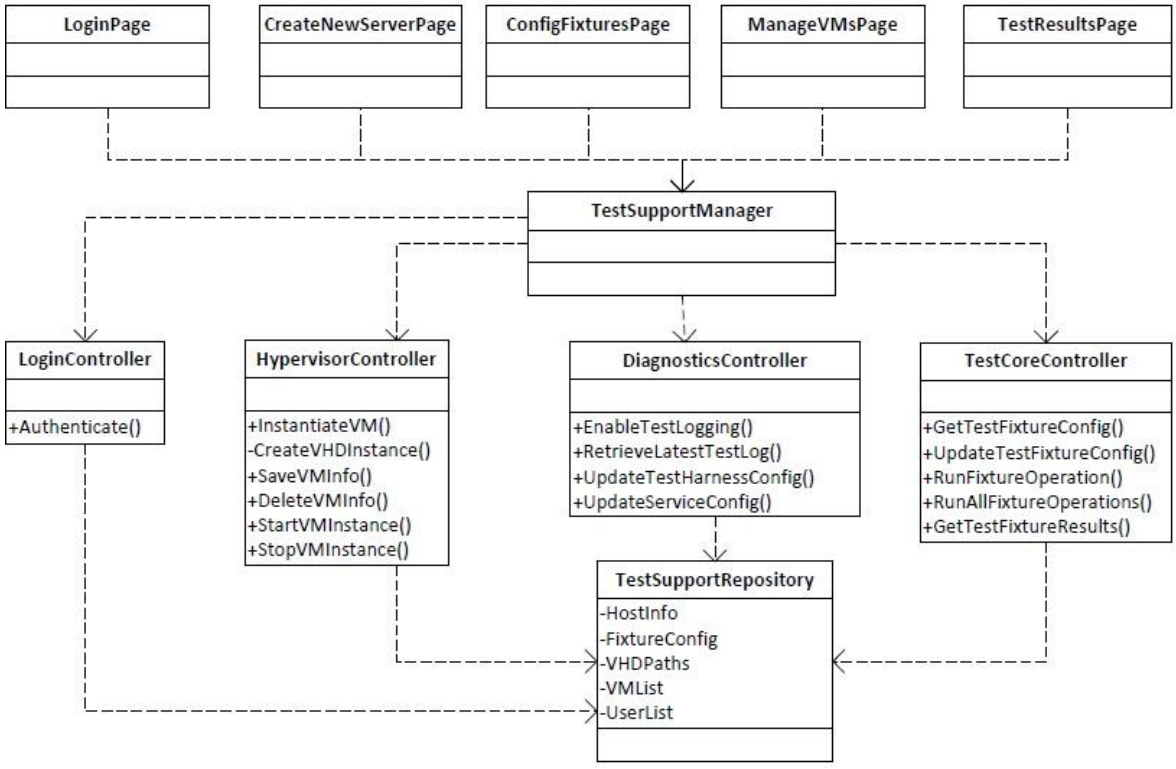

Figure 5. Class Diagram of the TSaaS Prototype

5.3.2. Dynamic Model

The Sequence Diagram in Figure 5.5 describes the behavior of the user interface prototype of TSaaS when the partner configures a virtual test server for the first time. The diagram assumes that the partner is logged into TSaaS, and has navigated to the Create Virtual Test Server Page.

The sequence of events, as relates to Figure 5.5, is as follows:

1. User fills the Create Virtual Test Server Page with information on the new virtual test server, and clicks the Create Button. This information includes a name for the server, the service to be hosted, and environment settings (i.e., the operating system, number of processors, and memory size).

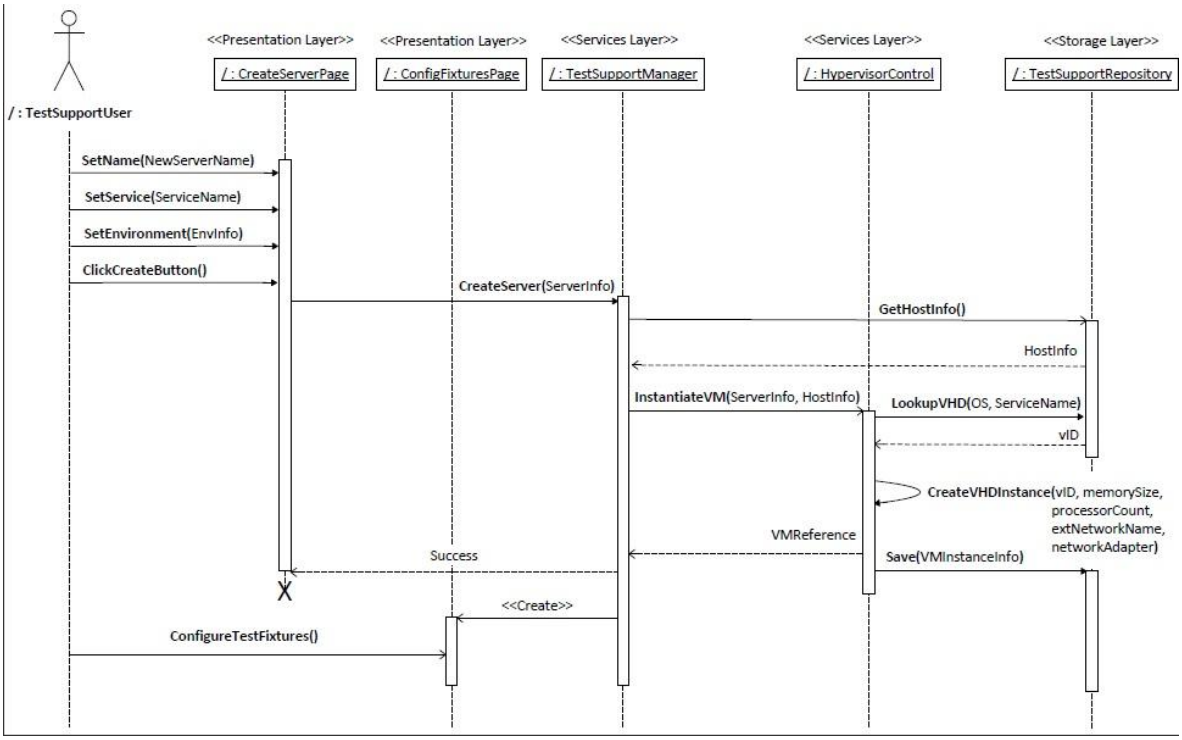

Figure 6. Dynamic Model for the TSaaS Prototype

2. A message to create a new test server with the given specifications is passed to the Test Support Manager (TSM). The TSM then makes a call to the Test Support Repository (TSR) to retrieve information on the physical machine where the server will be hosted. This includes the name of the host, its network adapter address, and the name of the external network.
3. The TSM requests that the HypervisorControl object instantiate the new virtual test server. This involves accessing the TSR to retrieve a virtual hard disk preconfigured with the requested OS and a test copy of the remotely-hosted service.
4. Once the new server has been instantiated, details on its configuration are stored in the TSR, and a reference to the server instance is returned to the TSM.
5. User is directed to Configure Test Fixtures Page (See Figure 5.3). This page allows the user to define and set up reusable test fixture objects that can be used in integration tests.

In the next chapter, I will explain the evaluation matrix, results and discussion to evaluate the TSaaS approach.

CHAPTER 6. EVALUATION

This chapter focuses on the evaluation of the Test Support as-a-Service (TSaaS). The matrix of evaluation table shown in this chapter will evaluate the approach in context of other cloud-based testing tools and other similar available approaches. Along with the evaluation matrix this chapter will also discuss the positives and negatives of the TSaaS approach and how TSaaS improves testability of cloud-based services.

6.1. Evaluation Matrix

Evaluation Parameters	TSaaS	D-Cloud	TaaS	Performance Tool (CloudTest by SOASTA)
Virtualization Support	√	√		
Data Security	√	√		√
Privacy	√			√
Availability	√			√
Integration testing	√			
Performance Testing				√
Regression Testing				
Functional Testing		√	√	√
Black Box Testing	√			
White Box Testing	√			
Error Reporting	√	√	√	√

Table 2: Evaluation Matrix

Table 2 shows the evaluation matrix to evaluate the TSaaS approach with other existing tools and approaches available. Several characteristics of the cloud computing environment have a negative impact on the testability of the application. The parameters I am using to evaluate the approach are a combination of the characteristics that have a negative impact on testability of application and different testing techniques. Following are the parameters used in the matrix to compare the TSaaS approach with other cloud testing tools and approaches.

1. Virtualization Support: One of the major benefits of virtualization in the software testing activity is convenience and cost savings through the use of fewer physical machines. Virtualization makes it easier to setup, execute, and teardown a variety of test scenarios using different resources and resource configuration.
2. Data Security: In the cloud computing environment services are hosted remotely rather than locally in the controlled environment to ensure that the potentially sensitive information which traverses the cloud is safe and secure.
3. Privacy: This parameter is important in order to ensure the privacy of the application users and associated information when used in Cloud.
4. Availability: Cloud computing must address the high dependability criteria such as availability.
5. Integration testing: Testing the integration of two cloud-based services is important to maintain the quality of the application.
6. Performance Testing: Performance tests should be performed to check performance criteria such as response time, memory usage and throughput.
7. Regression Testing: Allow user to perform regression to uncover new software bugs.
8. Functional Testing: Allow user to test the whole service.

9. Black Box Testing: Allow user to perform the testing without looking inside the service code.
10. White Box Testing: Allow user to test internal structure or working of the service code.
11. Error Reporting: Allow user to view and analyze the test results.

Following are the approaches and tools used to compare against the TSaaS approach in order to evaluate the approach:

D-Cloud: Takayuki Banzai et al. [14] propose D-cloud. D-Cloud is a software testing environment that uses cloud computing technology and virtual machines with fault injection facility. D-Cloud focuses testing of parallel and distributed systems in the real world after deployment, reliable systems, such as high-availability servers, are parallel and distributed systems. D-Cloud system allows a user to easily set up and test a distributed system on the cloud and effectively reduces the cost and time of testing.

TaaS: TaaS is an approach called “test as a service” proposed by George Candea et al. [15]. TaaS model provides automated software testing as a cloud-based service. They proposed three different kinds of TaaS: for developers to more thoroughly test their code, for end users to check the software they install, and certification services that enable consumers to choose among software products based on the products’ measured reliability. Unlike TSaaS, TaaS is dedicated to both technical and non-technical points of view of product. It helps consumers to make educated choices as well as enables developers to build better products.

CloudTest by SOASTA: SOASTA is a website testing service that includes load testing, software performance testing, functional testing and user interface testing [19]. SOASTA allows

users to use already defined tests or create customized tests to automatically test the web applications like TSaaS approach.

6.2. Result and Discussion

From the evaluation matrix shown in Table 2, we can conclude that the TSaaS approach offers better features as compared to the other existing approaches and tools. The proposed approach would be better if the missing features in the proposed approach were implemented. This section discusses about what we need to improve in TSaaS and what good the TSaaS approach already has in it.

Likely, TSaaS uses the built in virtualization power by cloud. The virtualization reduces the cost of the service and improves the testability of the application with the use of fewer physical machines and other virtualization benefits. From the evaluation matrix we can see that there is no tool available to test the integration of two cloud-based services. Hence TSaaS was found to be useful in the area of integration testing of two services. At the same time the TSaaS approach only supports integration testing. To improve the testability and efficiency of the service other types of testing such as performance and regression should be implemented as a part of this service to make a perfect cloud testing service. Also, TSaaS reuses the available test data, offers the partner a secure login and allows the partner to author new test cases and analyze test results.

The current version of TSaaS does not provide autonomic and adaptive features. Services in a cloud application can change independently of each other. A common reason for changing an individual service is for the purpose of software maintenance, and includes: fixing bugs in the service, adapting the service to a new environment, and extending the service to meet the

changing needs of clients. The current version of TSaaS cannot support performance testing and regression testing.

The TSaaS approach provides the user a protected interface to login to the system and hence the privacy of the application is maintained. The TSaaS approach uses the community cloud for its deployment, where the cloud is shared by multiple organizations having common interests and hence data security is achieved. Only the required data is made available to the user to achieve the high level of data security. With the virtualization support TSaaS maintains a controlled environment by creating a copy of the service to be tested and hence also increases the availability of the service being tested.

The next subsection highlights the positives and negatives of the TSaaS approach.

6.2.1. Benefits of TSaaS

This section highlights the benefits of Test Support as-a-Service (TSaaS) that will enlighten the usefulness of TSaaS to improve testability of cloud-based services:

1. TSaaS improves testability and confidence to the partner with secure login and controlled environment.
2. Provides improved product to provider as the partner can test provider's component and see bug report and fix bugs.
3. Virtualization benefits like snapshots that speeds up the testing process by bug replication, hardware savings, and environment testing (OS, Applications)
4. Allows the partner to integration test the service along with the high availability by maintaining the copy of the service.

6.2.2. Limitations of TSaaS

This section highlights the limitations of current version of Test Support as-a-Service (TSaaS) approach as per the evaluation:

1. The TSaaS approach cannot support performance testing and regression testing.
2. The TSaaS approach does not provide autonomic and adaptive functionalities. Hence, when every time there are changes in the service TSaaS has to update in order to match with the changes.
3. TSaaS needs standardization. Providing standardized interfaces and guidelines for performing testing in the cloud and developing test support infrastructures.

CHAPTER 7. CONCLUSION AND FUTURE WORK

In this paper, I designed the user interface for Test Support as-a-Service (TSaaS) approach to run the integration testing of the cloud-based services. I achieved this with the help of designing the interface with the help of architecture, detailed design and use case diagrams. This design will help the future research in the area of the TSaaS approach. I also implemented a small prototype that demonstrates the user interface design for the functionalities of TSaaS approach. This has demonstrated how a developer of the service would interact with TSaaS to test the other cloud-based services.

Furthermore, in this paper I also evaluated the TSaaS approach with the other existing approaches. I also discussed the evaluation parameters and the approaches that I have used to compare the TSaaS approach. The evaluation matrix helped me to figure out what are the positive and negative points of the TSaaS approach. The current version of proposed the TSaaS approach reuses test data and use the virtualization power of cloud that makes our approach more powerful as compared to the other available approaches. But at the same time its functionalities are limited to just the integration testing where the user is restricted to do only one type of testing. After observing the results from the evaluation matrix it is concluded that the TSaaS approach can be more effective and powerful if a support to test other types of testing were added.

Hence, the future work includes incorporation of performance-based testing techniques and automatic regression testing, into the TSaaS approach. This will make the approach more effective and powerful in future.

REFERENCES

[1] T. King and A. Ganti. Migrating autonomic self-testing to the cloud. In Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on, pages 438-443, April 2010.

[2] W. Chan, L. Mei, and Z. Zhang. Modeling and testing of cloud applications. In Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pages 111 - 118, December 2009.

[3] P. Mell and T. Grance. The nist definition of cloud computing (draft) recommendations of the national institute of standards and technology. Nist Special Publication, 145(6):7, 2011.

[4] T. R. Lab. Above the clouds: A Berkeley view of cloud computing. In Technical Report UCB/EECS-2009-28, EECS, Department, University of California, Berkeley, 2009.

[5] Wikipedia, Software-as-a-service, < http://en.wikipedia.org/wiki/Software_as_a_service >, retrieved on 2 February 2012.

[6] G. Inc. Google docs: Create and share your work online, <http://en.wikipedia.org/wiki/Google_Docs>, retrieved on 20 February 2012.

[7] M. Corporation. Microsoft office 365, <http://en.wikipedia.org/wiki/Microsoft_Office_365>, retrieved on 1 April 2012.

[8] Wikipedia. Platform-as-a-service, <http://en.wikipedia.org/wiki/Platform_as_a_service>, retrieved on 5 March 2012.

[9] M. Corporation, Microsoft windows azure, <http://en.wikipedia.org/wiki/Windows_Azure>, retrieved on 1 March 2012.

[10] Amazon.com. Amazon elastic computing cloud (amazon ec2), <http://aws.amazon.com/ec2/ > retrieved on 1 February 2012.

[11] Rackspace, < http://www.rackspace.com/>, retrieved on 5 February 2012.

[12] Wikipedia Gogrid, <http://en.wikipedia.org/wiki/GoGrid>, retrieved on 5 May 2012.

[13] Wikipedia Cloud testing, <http://en.wikipedia.org/wiki/Cloud_computing>, retrieved on 5 January 2012.

[14] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada,T. Hanawa, and M. Sato. D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID '10, IEEE Computer Society, pages 631-636, Washington, DC, USA, 2010.

[15] G. Candea, S. Bucur, and C. Zamfir. Automated software testing as a service. In Proceedings of the 1st ACM symposium on Cloud computing, SoCC '10, pages 155-160, ACM, New York, NY, USA, 2010.

[16] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea. Cloud9: a software testing service. SIGOPS Oper. Syst. Rev., 43:5-10, January 2010.

[17] M. Greiler, H.-G. Gross, and A. van Deursen. Evaluation of online testing for services: a case study. In Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems, PESOS '10, pages 36-42, New York, NY, USA, 2010.

[18] T. M. King, A. S. Ganti, and D. Froslie. Enabling automated integration testing of cloud application services in virtualized environments. In Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '11, IBM Corp, pages 120-132, Riverton, NJ, USA, 2011.

[19] SOASTA, Cloud Testing Service, < http://en.wikipedia.org/wiki/SOASTA>, retrieved on 2 March 2012.

APPENDIX A. SCREEN SHOTS OF TSAAS

A.1. Authentication

The figure shows the screenshot for Authentication process of the TSaaS (Test Support as-a-Service) user interface prototype. From here the partner can login to the TSaaS with valid the user name and password. In this way the privacy of the service is maintained over the cloud by TSaaS.

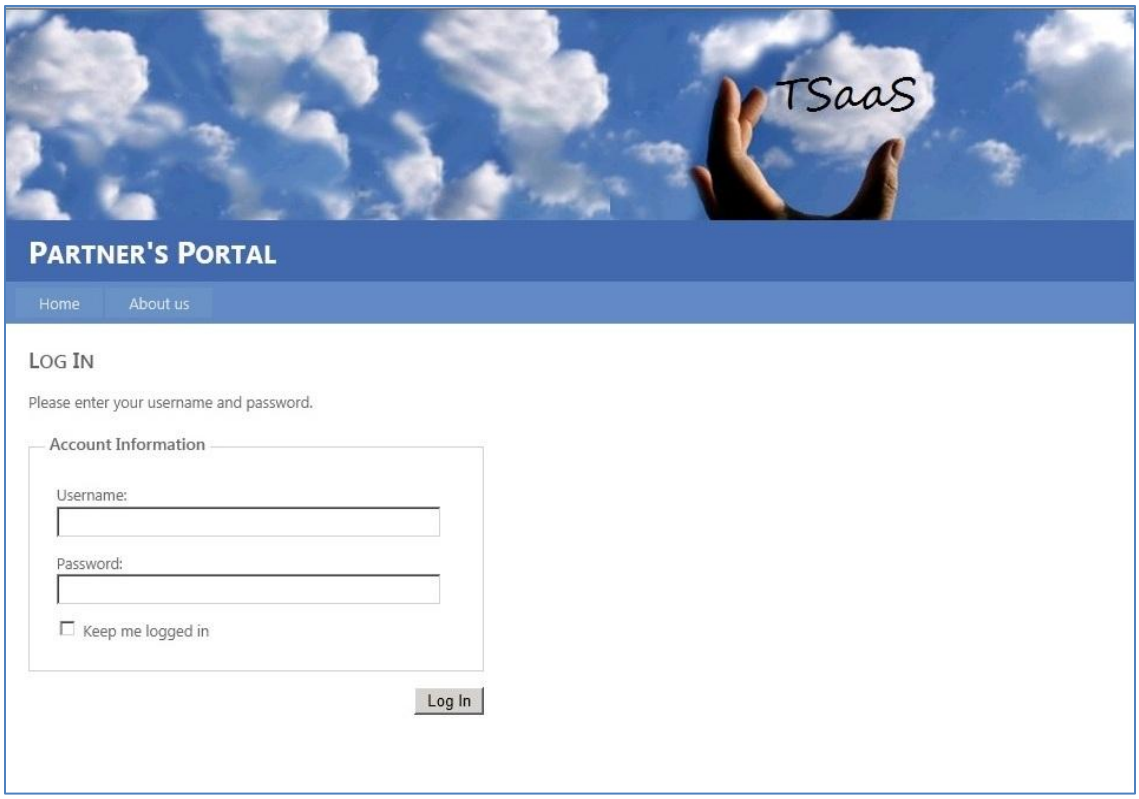

Figure A1. Authentication

A.2. Runtime Virtualization I

The figure shows the screenshot of the TSaaS user interface prototype explaining the runtime virtualization process. From here the partner will be able to select the service to test. Also, the partner can select the virtual test server to run the tests, so that tests would run without disturbing the normal operation of the service to test. If a virtual test server is not available the partner can create one of his choices to run the tests to the fullest.

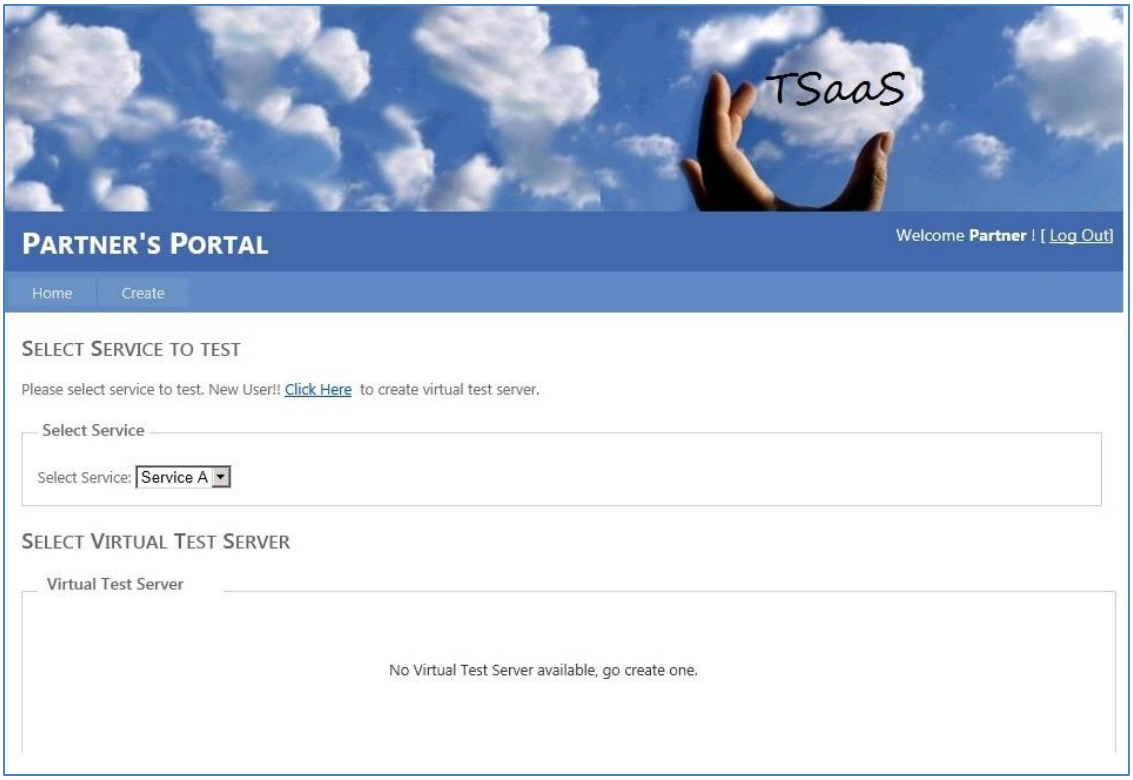

Figure A2. Runtime Virtualization

A.3. Configure Virtual Test Environment

The figure shows the screenshot of the TSaaS user interface prototype to configure virtual test environment of the partner's choice. From the above screen the partner will be able to create the virtual test server for the testing purpose. This part of application uses the built-in cloud computing technology, virtualization. To improve the process of testing cloud- based application.

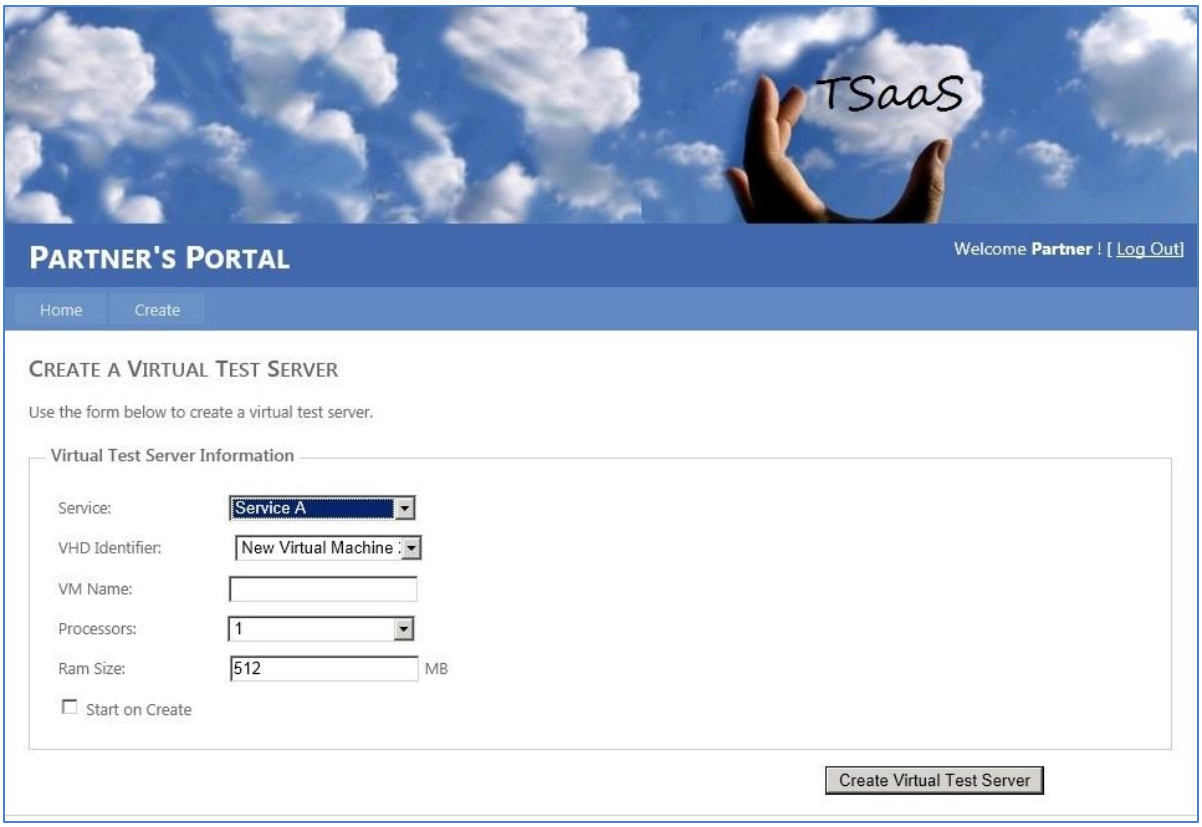

Figure A3. Configure Virtual Test Server

A.4. Runtime Virtualization II

The figure shows the screenshot of the TSaaS user interface prototype where the partner can perform operations related to virtualization – create, start, stop, delete, take snapshot, and update server configuration.

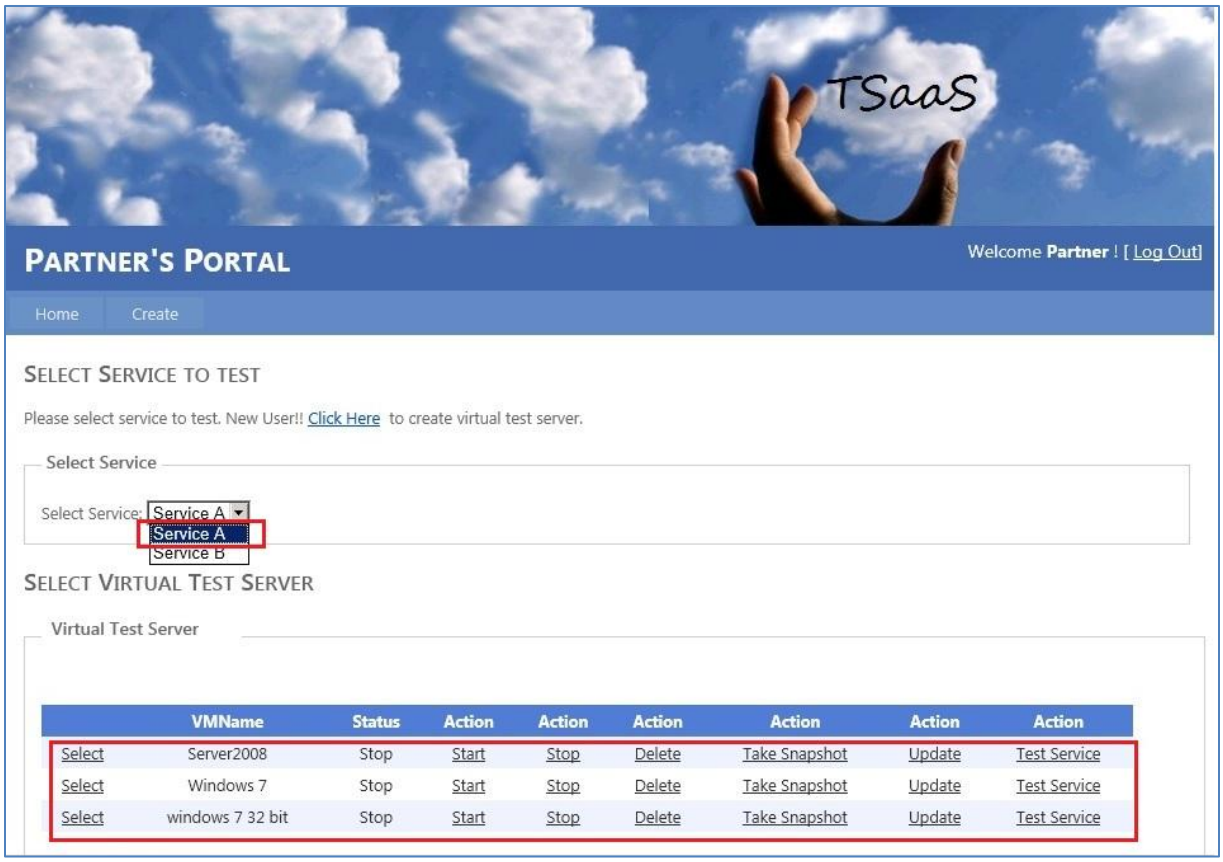

Figure A4. Runtime Virtualization II

A.5. Test Specification & Test Execution

The figure shows the screenshot of the TSaaS user interface prototype where the partner can test the application either by selecting tests in its original form or can create the customized test cases by selecting Edit.

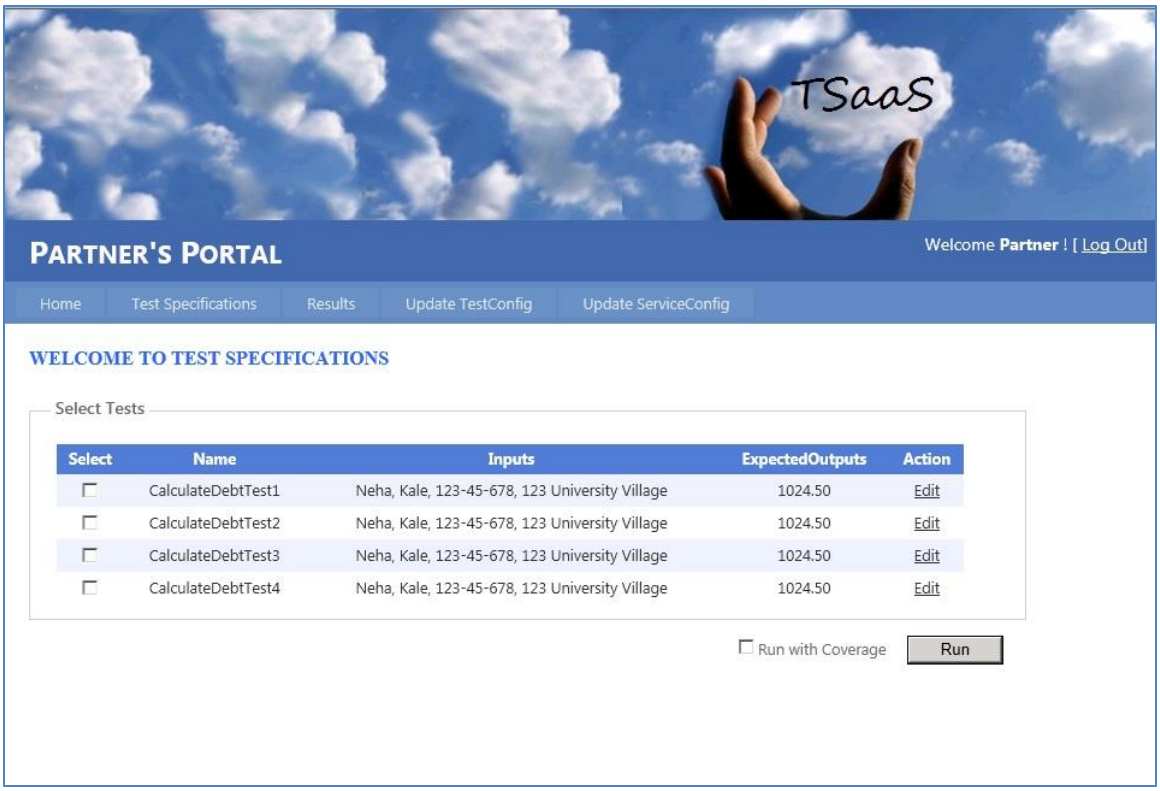

Figure A5. Run Tests

A.6. Test Execution Result

The figure shows the screenshot of the TSaaS user interface prototype where the partner can see the test execution results. The partner can also see the details in order to analyze the failure.

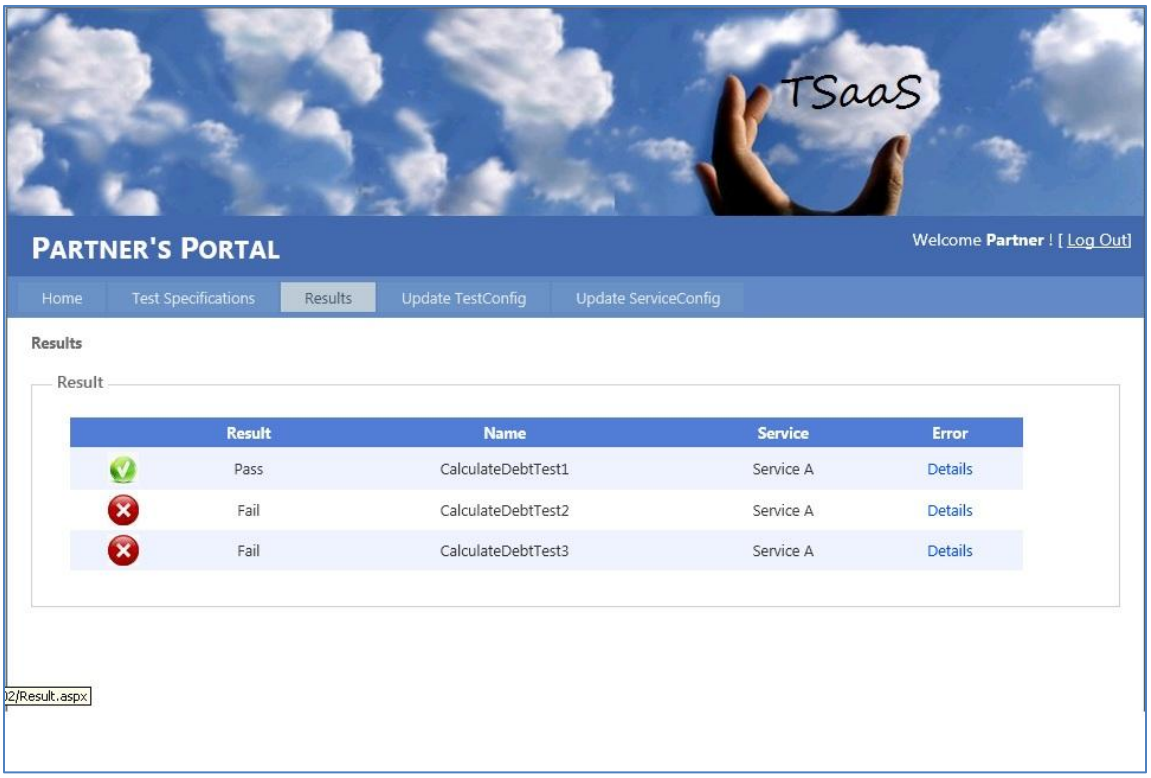

Figure A6. Test Results

APPENDIX B. USE CASE SCENARIOS

Use Case ID: TSaaSUC01_PartnerAuthentication

Use Case Level: End-toEnd

***Details:** This use case helps partner to get authorized access to TSaaS

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. Open any web browser.
2. Enters the address of Test Support as a Service (TSaaS) in the web browser.

- **Description:**

1. Use case begins when Partner successfully directed to The TSaaS home screen.
2. The service shall provide Partner with a template for data entry (Figure A1.Authentication).
3. The Partner shall enter correct Partner name.
4. The Partner shall enter correct Password.
5. The Partner shall then click to Log In.
6. Use case ends when the Partner is redirected to the next screen successfully.

- **Post-conditions:**

1. Partner shall be successfully login to the TSaaS
2. Partner shall be directed to the next screen to run service. (Figure A2. Runtime Virtualization)

***Alternative Courses of Action:** None.

***Exceptions:** None

1. Partner enters wrong Log In information.

***Related Use Cases:** None

Decision Support

***Frequency:** On average 10 requests are made daily by partner.

***Criticality:** High. Allows the Partner to Log In to the service.

***Risk**: High. Implementing this use case helps Partner to securely Log In to the service and helps service to retrieve and set test environment for the Partner.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 7/9/2011

***Date last modified:** 14/12/2011

***Use Case ID:** TSaaSUC02_Create Virtual Test Server

Use Case Level: End-toEnd

***Details:** This use case helps Partner to create virtual test server to test service.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS

2. Partner logged to TSaaS in using predefined credentials and is valid.

- **Description:**

1. Use case begins when partner is directed to runtime virtualization screen after successful validation. (Figure A2. Runtime Virtualization)
2. The new partner must click to "Click Here" to create a new virtual test server.
3. The Test Support as a Service (TSaaS) shall take the partner to create new virtual test server screen. (Figure A3. Virtual Test Server)
4. The TSaaS shall provide partner a template to create virtual test server for testing a service under test. (Figure A3. Virtual Test Server)
5. The partner shall select service to test.
6. The partner shall select Virtual Hard Disk (VHD) Identifier to select VHD create virtual test server.
7. The partner shall select the path of virtual hard disk.
8. The partner shall enter Virtual Test Server name.
9. The partner shall enter name of processor.
10. The partner shall enter RAM Size.
11. The partner shall enter Virtual Hard disk path
12. The partner may select "Start on Create" to run virtual test server on create.
13. Partner shall click "Create Virtual Test Server" to create new virtual test server.
14. Use case ends when the TSaaS notifies partner that new virtual test server is created

- **Relevant requirements:** None.

- **Post-conditions:**

1. The newly created virtual test server is added to runtime virtualization screen.

2. Newly created virtual test server is created in the Hyper-V Manager and status shall be running.

***Alternative Courses of Action**

1. The partner may not select "Start on Create" to run virtual test server on create.

- **Post-conditions:**

1. The newly created virtual test server is added to runtime virtualization screen.
2. Newly created virtual test server is created in the Hyper-V Manager and status shall be stop.

***Exceptions:** Any of the given fields are null.

Concurrent Uses: None.

***Related Use Cases:** None

Decision Support

***Frequency:** High: Every new partner has to create a virtual test server.

***Criticality**: High: This is very important for the partner to be able to create virtual test server in order to test service

***Risk**: High: The partner will not be able create server and test service if this test case is not implemented.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 7/9/2011

***Date last modified:** 14/12/2011

***Use Case ID:** TSaaSUC03_Start Virtual Test Server

Use Case Level: End-toEnd

***Details:** This use case helps partner to start virtual test server to test selected service.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. The selected virtual test server exist at Hyper-V Manager.
4. The selected virtual test server is not running.

- **Description:**

1. Use case begins when partner is directed to runtime virtualization screen after successful validation. (Figure A2. Runtime Virtualization)
2. The TSaaS shall provide partner a template for runtime virtualization. (Figure A4. Runtime Virtualization).
3. The partner shall select service to test.
4. The TSaaS shall display the available test servers for the selected service.
5. The partner shall select the required test server from the list of available test servers.
6. The partner shall click to "Start" to start the selected virtual test server.
7. Use case ends when the TSaaS notifies partner that the server started successfully.

- **Relevant requirements:** None.

- **Post-conditions:** The selected test server is started to test the selected service.

***Alternative Courses of Action:** None.

Extensions: None

***Exceptions:**

1. The selected virtual test server does not exist.
2. The selected virtual test server to start is already running.

Concurrent Uses: None.

***Related Use Cases**: TSaaSUC04_Stop Virtual Test Server to stop the selected virtual test server and TSaaSUC05_Delete Virtual Test Server to delete the selected virtual test server.

Decision Support

***Frequency**: High: Every time to test the selected service partner has to start the virtual test server.

***Criticality**: High: this is very important for the partner to be able to start sever in order to test the service under test.

***Risk:** High: The partner will not be able to start the virtual test server and perhaps not be able to test the service.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 7/9/2011

***Date last modified:** 14/12/2011

***Use Case ID:** TSaaSUC04_Stop Virtual Test Server

Use Case Level: End-toEnd

***Details:** This use case helps partner to stop virtual test server.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. The selected virtual test server exist at Hyper-V Manager.
4. The selected virtual test server is running.

- **Description:**

1. Use case begins when partner is directed to runtime virtualization screen after successful validation. (Figure A2. Runtime Virtualization)
2. The TSaaS shall provide partner a template for runtime virtualization. (Figure A4. Runtime Virtualization).
3. The partner shall select service to test.
4. The TSaaS shall display the available test servers for the selected service.
5. The partner shall select the required test server from the list of available test servers.
6. The partner shall click to “Stop” to stop the selected virtual test server.

7. Use case ends when the TSaaS notifies partner that the virtual test server stopped successfully.

- **Relevant requirements:** None.
- **Post-conditions:** The selected virtual test server is stopped.

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:**

1. The selected virtual test server does not exist.
2. The selected virtual test server to stop is already at stop state.

Concurrent Uses: None.

***Related Use Cases:** TSaaSUC03_Start Virtual Test Server to start the selected virtual test server and TSaaSUC05_Delete Virtual Test Server to delete the selected virtual test server.

Decision Support

***Frequency:** Low: Partner may need to stop the virtual test server.

***Criticality:** Medium: This may be necessary for the partner to be able to stop the running virtual test server.

Risk: Medium: The partner will not be able to stop the virtual test server.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 7/9/2011

***Date last modified:** 14/12/2011

***Use Case ID:** TSaaSUC05_Delete Virtual Test Server

Use Case Level: End-toEnd

***Details:** This test case helps partner to delete the virtual test server.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. The selected virtual test server exist at Hyper-V Manager.

- **Description:**

1. Use case begins when partner is directed to runtime virtualization screen after successful validation. (Figure A2. Runtime Virtualization)
2. The TSaaS shall provide partner a template for runtime virtualization. (Figure A4. Runtime Virtualization).
3. The partner shall select service to test.
4. The TSaaS shall display the available test servers for the selected service.
5. The partner shall select the required test server from the list of available test servers.
6. The partner shall click to “Delete” to delete the selected virtual test server
7. Use case ends when the TSaaS notifies partner that the virtual test server deleted successfully.

- **Relevant requirements:** None.
- **Post-conditions:** The selected test server is deleted from Hyper-V Manager.

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:**

1. The selected virtual test server does not exist.

Concurrent Uses: None.

***Related Use Cases:** TSaaSUC03_Start Virtual Test Server to start the selected virtual test server and TSaaSUC04_Stop Virtual Test Server to stop the the selected virtual test server.

Decision Support

***Frequency:** Medium: Partner may need to delete the virtual test server.

***Criticality:** Medium: This may be necessary for the partner to be able to delete virtual test server.

***Risk:** Medium: The partner will not be able to delete the virtual test server.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 2/1/2011

***Date last modified:** 15/12/2011

***Use Case ID:** TSaaSUC06_Update Virtual Test Server

Use Case Level: End-toEnd

***Details:** This test case helps partner to update the selected virtual test server.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. The selected virtual test server exist at Hyper-V Manager.
4. The selected virtual test server is not running.

- **Description:**

1. Use case begins when partner is directed to runtime virtualization screen after successful validation. (Figure A2. Runtime Virtualization)
2. The TSaaS shall provide partner a template for runtime virtualization.(Figure A4. Runtime Virtualization).
3. The partner shall select service to test.
4. The TSaaS shall display the available test servers for the selected service.
5. The partner shall select the required test server from the list of available test servers.
6. The partner shall click to "Update" to update the selected virtual test server.
7. The TSaaS shall direct the partner to the update screen.
8. The partner shall edit RAM size and/or processor for the selected virtual test server.

9. The partner shall click “Update” to save changes.

10. Use case ends when the TSaaS notifies partner that the virtual test server updated successfully.

- **Relevant requirements:** None.

- **Post-conditions:** The selected test server is updated successfully.

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:**

11. The selected virtual test server does not exist.

12. The selected virtual test server is running.

Concurrent Uses: None.

***Related Use Cases:** TSaaSUC03_Start Virtual Test Server to start the selected virtual test server and TSaaSUC04_Stop Virtual Test Server to stop the the selected virtual test server. TSaaSUC05_Delete Virtual Test Server to delete virtual test server.

Decision Support

***Frequency:** Medium: Partner may need to delete the virtual test server.

***Criticality:** Medium: This may be necessary for the partner to be able to delete virtual test server.

***Risk:** Medium: The partner will not be able to delete the virtual test server.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 15/12/2011

***Date last modified:** -

***Use Case ID**: TSaaSUC07_RunAllTests

Use Case Level: End-toEnd

***Details:** This use case helps partner to select and run all tests to test the service.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. Service to test is slected.
4. Virtual server to run tests is selected.

- **Description:**

1. Use case begins when partner clicks to "Test Service" from runtime virtualization screen of selected virtual test server. (Figure A4. Runtime Virtualization)
2. The TSaaS shall take the partner to the Test Service screen. (Figure A5. Run Tests)
3. The TSaaS shall provide partner a template where partner shall see Virtual server information and be able to select tests to run. (Figure A5. Run Tests)

4. The partner shall select all tests.
5. The partner shall click to “Run”.
6. Until the tests are running the TSaaS must show partner the progress bar to show the status of running tests.
7. Use case ends when all the test completes their excecution and TSaaS shall direct partner to the results screen. (Figure A6. Test Results)

- **Relevant requirements:** None.
- **Post-conditions:** Results screen shall be displayed to the partner where partner will be able to see test results

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:** None.

Concurrent Uses: None.

***Related Use Cases:** TSaaSUC08_RunTests partner can run selected tests to test the service and TSaaSUC09_RunAllTestswithCodeCoverage to run tests with code coverage .

Decision Support

***Frequency:** High: Partner shall always need to run tests to test service.

***Criticality:** High: This is very important for partner to run tests in order to test the service.

***Risk:** High: The partner will not be able to run tests if this use cases is not implemented.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 13/10/2011

***Date last modified:** 15/12/2011

***Use Case ID**: TSaaSUC08_RunTests

Use Case Level: End-toEnd

***Details:** This use case helps partner to select and run slected tests to test the service

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. Service to test is slected.
4. Virtual server to run tests is selected.

- **Description:**

1. Use case begins when partner clicks to "Test Service" from runtime virtualization screen of selected virtual test server. (Figure 4 Appendix A: Runtime Virtualization)
2. The TSaaS shall take the partner to the Test Service screen. (Figure 5 Appendix A: Run Tests)

3. The TSaaS shall provide partner a template where partner shall see Virtual server information and be able to select tests to run. (Figure 5 Appendix A: Run Tests)
4. The partner shall select required tests to run.
5. The partner shall click to “Run”.
6. Until the tests are running the TSaaS must show partner the progress bar to show the status of running tests.
7. Use case ends when all the selected tests complete their excecution and TSaaS shall direct partner to the results screen. (Figure 6 Appendix A: Test Results)

- **Relevant requirements:** None.
- **Post-conditions:** Results screen shall be displayed to the partner where partner will be able to see test results

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:** None.

Concurrent Uses: None.

***Related Use Cases:** TSaaSUC06_RunAllTests to run all tests to test service and TSaaSUC08_RunAllTestswithCodeCoverage to run tests with code coverage .

Decision Support

***Frequency:** High: Partner shall always need to run tests to test service.

***Criticality:** High: This is very important for partner to run tests in order to test the service.

***Risk:** High: The partner will not be able to run tests if this use cases is not implemented.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 13/10/2011

***Date last modified:** 15/12/2011

***Use Case ID**: TSaaSUC09_RunAllTestswithCodeCoverage

Use Case Level: End-toEnd

***Details:** This use case helps partner to select and run all test to test the service

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. Service to test is slected.
4. Virtual server to run tests is selected.

- **Description:**

1. Use case begins when partner clicks to "Test Service" from runtime virtualization screen of selected virtual test server. (Figure A4. Runtime Virtualization)
2. The TSaaS shall take the partner to the Test Service screen. (Figure A5. Run Tests)

3. The TSaaS shall provide partner a template where partner shall see Virtual server information and be able to select tests to run. (Figure A5. Run Tests)
4. The partner shall select the test cases to run.
5. The partner must select to “Run with CodeCoverage”.
6. Until the tests are running the TSaaS must show partner the progress bar to show the status of running tests.
7. Use case ends when all the test completes their excecution and TSaaS shall direct partner to the results screen. (Figure A6. Test Results)

- **Relevant requirements:** None.
- **Post-conditions:** Results screen shall be displayed to the partner where partner will be able to see test results with code coverage.

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:** None.

Concurrent Uses: None.

***Related Use Cases:** TSaaSUC06_RunAllTests to run all tests to test service and TSaaSUC07_RunTests partner can run selected tests to test the service.

Decision Support

***Frequency:** High: Partner shall always need to run tests to test service.

***Criticality:** High: This is very important for partner to run tests in order to test the service.

***Risk:** High: The partner will not be able to run tests if this use cases is not implemented.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 13/10/2011

***Date last modified:** 15/12/2011

***Use Case ID**: TSaaSUC10_UpdateTests

Use Case Level: End-toEnd

***Details:** This use case helps partner to update tests to test the service throughlly.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS
2. Log in using predefined credentials and is valid
3. Service to test is slected.
4. Virtual server to run tests is selected.

- **Description:**

1. Use case begins when partner clicks to "Edit" from Test Specificatuion and Exceution screen. (Figure A5. Run Tests).

2. The TSaaS shall display partner a screen where partner shall update tests. (Figure A5. Run Tests).
3. Partner shall select the required test case to update.
4. Partner shall update the test case.
5. Partner shall click “Update” to save the updates.
6. Use case ends when the updates gets saved to the test successfully.

- **Relevant requirements:** None.
- **Post-conditions:** All updates shall be saved to TSX file.

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:** None.

Concurrent Uses: None.

***Related Use Cases:** None

Decision Support

***Frequency:** High:

***Criticality:** High: This is very important for the user to update the basic tests to test the funtionality of the service under tests throughlly.

***Risk:** High: The partner will not be able to update the test and hence not be able to test the service throughlly.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 13/10/2011

***Date last modified:** 15/12/2011

***Use Case ID**: TSaaSUC11_GetLatestTestResults

Use Case Level: End-toEnd

***Details:** This use case helps partner to get latest test results.

- **Actor:** Partner– Person using TSaaS
- **Pre-conditions:**

1. A Partner provider connected to TSaaS.
2. Logged in using predefined credentials and is valid.
3. Service to test is slected.
4. Virtual server to run tests is selected.

- **Description:**

1. Use case begins when partner clicks to "Test Service" from runtime virtualization screen of selected virtual test server. (Figure A4. Runtime Virtualization Screen)
2. The TSaaS shall take the partner to the Test Service screen. (Figure A5. Run Tests)
3. The TSaaS shall provide partner a template where partner shall see Virtual server information and be able to select tests to run. (Figure A5. Run Tests)

4. The partner shall select tests to run.

5. The partner shall click “Run” to run the selected tests.

6. The TSaaS shall display progress bar to show the progress of the running tests.

7. Use case ends when TSaaS shall display the results screen to the partner. (Figure A6. Test Results)

- **Relevant requirements:** None.

- **Post-conditions:** None

***Alternative Courses of Action:** None.

Extensions: None.

***Exceptions:** None.

Concurrent Uses: None.

***Related Use Cases:** None

Decision Support

***Frequency:** High: Required frequently.

***Criticality:** High: This is very important for user to see the results of the tests.

***Risk:** High: The partner will not be able to see the result of the service under test.

Modification History --

***Owner:** Neha Kale

***Initiation date:** 13/10/2011

***Date last modified:** 15/12/2011