Testing Cloud Services Using the TestCast Tool

101

Stelios Sotiriadis, Andrus Lehmets, Euripides G.M. Petrakis, and Nik Bessis

Abstract
This work presents the testing requirements for cloud services including unit and integration testing by identifying services that could communicate with each other according to their APIs. We also present the Elvior TestCast T3 (TTCN-3) testing tool that provides an efficient and easy to use solution for automating functional tests. This allows incremental development where users can test specific systems and features separately as well as the entire system as a whole. We finally demonstrate the empirical results as lessons learned from our experiences when apploying such solution in testing real world cloud health services.

Keywords
Cloud computing • Cloud services • Cloud service testing • TestCast • TTCN-3

101.1 Introduction

Cloud computing offers on demand services over the Internet to users and developers that combine easy access to computing resources with remote data management, elasticity, self service provisioning (allowing users to set-up and launch applications as services) and certain economic benefits. It supports different models to cover a variety of cloud users', such as the Infrastructure, Platform and Software as a Service (IaaS, PaaS and SaaS). The evolution of these, characterizes the so-called Future Internet (FI) concept [1] and allows development of innovative applications from modular services referred to as cloud enablers. In particular, developers build applications by utilizing on-the-self services encapsulating common functionalities (e.g. user authentication, data storage, context data management etc.), instead of re-engineering and implementing services from scratch.

Another important aspect of these services is the modularity that in cloud computing is enabled by deploying SaaS whose specifications are open and are available for utilization using APIs (i.e. as RESTFul interfaces [2]). While these services highlight innovation and promotion of a new easy-going development method, software engineering processes are becoming more and more complex. This is because such enablers have distinct features that impact several different research fields, including software testing [3]. These include execution over virtualized resources [5] that can be highly scalable and elastic, for instance, the users can increase their computational capacities and share

S. Sotiriadis (✉)
Computer Engineering Research Group, University of Toronto, Toronto, Canada

School of Electronic and Computer Engineering, Technical University of Crete, Chania, Greece
e-mail: s.sotiriadis@utoronto.ca

A. Lehmets
Elvior, Tallinn, Estonia
e-mail: andrus.lehtmets@elvior.ee

E.G.M. Petrakis
School of Electronic and Computer Engineering, Technical University of Crete, Chania, Greece
e-mail: petrakis@intelligence.tuc.grl

N. Bessis
Edge Hill Univerity, Ormskirk, UK
e-mail: Nik.Bessis@edgehill.ac.uk

common physical resources with other cloud applications or services, thus supporting multi-tenancy. Also such services provide interfaces using APIs that allow their combination to form composite services (e.g. services developed by extending or combining others).

Cloud services' testing introduces a new set of challenges and requirements [4] that have a direct impact on system engineering processes. The testing methodology involves not only validation of a software processes, but also validation of services APIs and of their interactions with other services. This involves testing of the actual service interactions based on their input and output interfaces. The complexity factor is also increased due to the fact that software engineers base their design on abstract and high-level use case models. Thus it becomes a hurdle to design test cases that are based (a) on the application requirements e.g. expecting behaviour of the services according to a use cases and (b) on the services it self (that is related with the functionalities of the cloud enablers).

In this work we present a testing methodology for testing cloud services utilizing the TestCast TTCN3 by Elvior.[1] The tool provides a programming language for testing communication protocols and Web services in an easy and efficient way. Based on this, Sect. 101.2 presents the motivation and background for this work, Sect. 101.3 the proposed testing methodology of cloud services, Sect. 101.4 the analysis of the tool, Sect. 101.5 the lessons learned by applying the tool in real world use cases and in Sect. 101.6 the conclusions and future research directions.

101.2 Motivation and Background

This work is based on the Future Internet Social and Technological Alignment Research (FI-STAR)[2] FP7 project that attempted to establish early trials in the health domain and in the context of FI-PPP EU initiative.[3] FI-STAR purpose was to prepare industry take-up of the developed cloud technology. It also build upon cloud services called Generic Enablers (GEs) provided by FIWARE.[4] GEs are software modules that offer various functionalities along with protocols and interfaces for operation and communication. These include the cloud management for supervision of the underlying infrastructure, the utilization of various IoT devices for data collection, tools for data analytics and communication interfaces for gateways and end-users. All FIWARE GEs are stored in a public catalogue, thus developers could easily browse and select appropriate APIs to use. As a side-result, FI-STAR aims to create a framework (a software to data approach) to allow GEs to be delivered to different physical locations.

Today, there are new requirements related to the software engineering processes of cloud enablers since there are significant changes taking place as to how to test the new software running on the latest cloud platforms. Especially in the case of software testing, this refers to the gap between software developers and software test engineers. There are three key differentiations among traditional and cloud enablers testing and these are related with (a) scalability of virtual resources, (b) multi-tenancy efficiency that refers concurrent users and (c) dynamic reconfiguration of services. As a result, the cloud testing models have to support different kinds of requirements [14], thus, tend to become more and more complex. In more detail, traditional applications are firstly designed and then tested. This work is motivated by work of [15, 16] where authors suggest that there is a lack of research papers addressing new issues, challenges, and needs in SaaS testing. In [16] we present the details of the unit and integration testing methodology for future Internet cloud services. We focus on cloud services testing and we propose a methodology enabling efficient unit and integration testing of modular services.

101.3 Testing Methodology of Cloud Services

The methodology includes the testing requirements analysis and the test cases analysis of cloud enablers. The testing preparation strategy defines the testing schedule and plan for black box testing of cloud enablers including conceptualization for unit and integration testing. Firstly, the unit testing allows cloud services' testers to execute various tests with different input parameters to the operations and interfaces defined in service specification documents and manuals. The modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are properly developed. The integration testing will allow combination and test execution of linked cloud enablers in order to test key functionalities derived by application use-cases.

The testing starts with the unit testing preparation activity where each cloud enabler tester defines the unit test and unit test cases so the unit test defect management process will allow bug tracking and fixing. Then the integration testing includes preparation, execution and defection management process regarding the inter-dependencies of enablers. At this stage, the aim is to exploit the building blocks of the application that map to the key functionalities. In more detail, the unit testing includes the testing activity of the cloud enablers by focusing on the evaluating of their interfaces. Integration

[1] http://www.elvior.com/testcast/ttcn-3.
[2] https://www.fi-star.eu.
[3] https://www.fi-ppp.eu.
[4] https://www.fiware.org.

testing includes exploration of the interactions between the services, such as their commincation and their input-output bonds. Next we focus on the TestCast TTCN-3 tool.

101.4 General overview of the TestCast TTCN-3 Tool

The Elvior TestCast is a full featured TTCN-3 tool (a programming language for testing of communication protocols and Web services) for automated testing of cloud systems. TestCast is ideal for incremental development where users can test specific systems and features separately as well as the entire system as a whole. Figure 101.1 demonstrates the Principal architecture of TTCN-3 test environment.

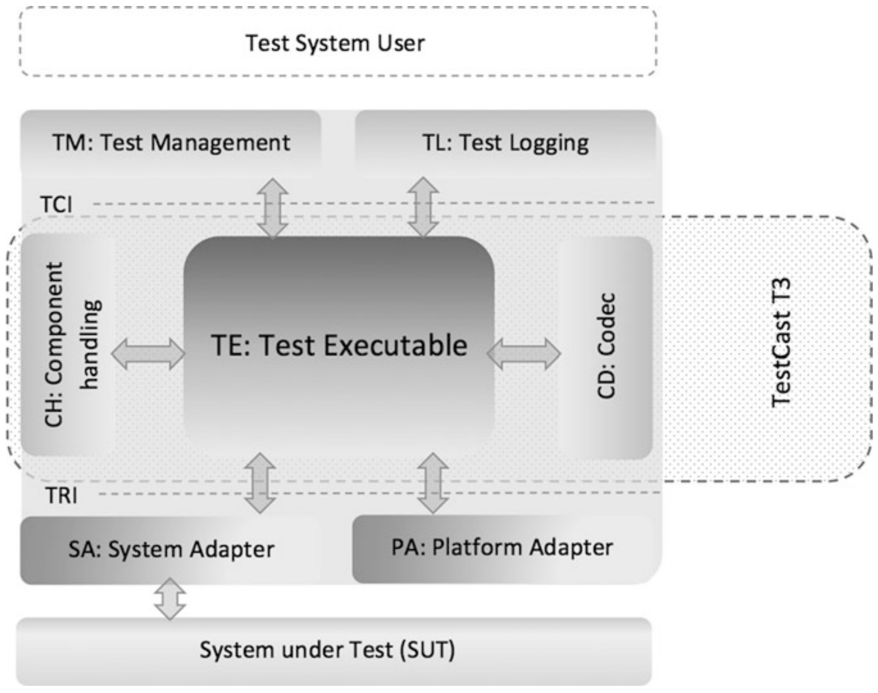
In detail, the System Adapter (SA) is used for communication of test tool (test executable) with System Under Test (SUT). The interface between test tool and SA is standardized and called TRI (TTCN-3 Runtime Interface) – SA is usually a piece of software that can be written in different languages such as C, C++, C#, Java, mainly depending on test tool. The other important interface is TCI (TTCN-3 Control Interface) is essential part of TTCN-3 test environments because its type system is not bind to any binary representation. It is entirely up to the test tool and its codecs to ensure encoding and decoding of data in appropriate format.

The characteristics of the Elvior TTCN-3 tool are as follows. The tool is developed in C# and could be run in Microsoft Windows and Linux platforms. Also, it has a user-friendly graphical interface for test development and management. It has a native compiler that supports all TTCN-3 standards (up to TTCN-3:2015) and includes a native TTCN-3 debugger. The tool supports TTCN-3 test execution and XSD import. It has built in codecs such as textual, binary, XML (via XSD schemas), TCI XML, ASN.1 and supports TRI and TCI mapping for C, C++, C# and Java. Finally, it offers a rich TTCN-3 editor and test suite viewer and a enriched logging (textual and graphical views) and logs analyzing capabilities. The tool provides a test environment for RESTful interfaces testing. The TTCN-3 based test environment architecture for functional Black-Box testing (BBT) includes an HTTP System Adapter, XML and JSON codecs and SUT (implementation using RESTful API). The roles of actors shown in Fig. 101.2 in the TestCast Tool testing environment are presented bellow.

(a) System Under Test (SUT): The implementation using the RESTful API.
(b) TestCast T3 test tool for TTCN-3 test development and execution (it is responsible for execution of TTCN-3 test scripts).
(c) HTTP SA (developed in C#): It completes communication between SUT and test tool. It acts as a server or client depending on test case needs and is controlled by TTCN-3 test script.
(d) XML TestCast T3 built in codec and the external JSON codec: Codecs are responsible for encoding and decoding messages sent/received to/from SUT.

In particular, for RESTful interfaces a generic TTCN-3 framework for sending and receiving HTTP messages was created. The scripts are written and executed in TestCast T3,

Fig. 101.1 Principal architecture of TTCN-3 test environment



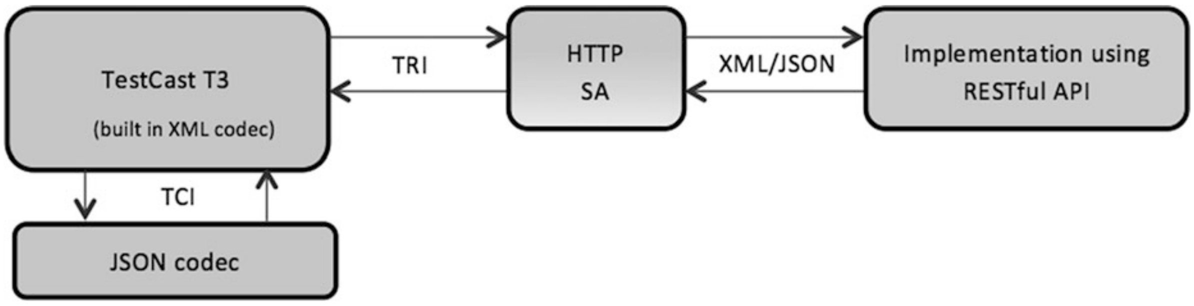


Fig. 101.2 Architecture of BBT environment for RESTful APIs

which sends messages to the HTTP adapter. The System Adapter creates HTTP messages based on the received information from TestCast and sends each of which to the SUT. In practice, the following actions are executed: i) message (hex sequence) is received from the SUT via SA to TestCast, ii) TestCast (TTCN-3 test executable) reads this hex sequence from TTCN-3 port and iii) the message is decoded by TestCast and can be used in TTCN-3 script.

It should be mentioned that TTCN-3 libraries include common types and templates for forming messages for RESTFul APIs. These common types and templates are used for creating implementation specific messages and respective test cases.

101.5 Lessons Learned

This section presents the experiences regarding the "lessons learned" from the unit and integration testing of cloud enablers when applied in the FI-STAR project. During this work we focused on the following research contributions.

(a) How to perform testing (regarding unit and integration) of cloud enablers? We presented a methodology that encompasses unit (white and black box) and integration testing (top down and bottom up) of cloud enablers. The cloud enablers tested are decentralized and composed by 3rd party services based on the real world case of FI-STAR.
(b) How to provision decision support to the cloud application developers during the testing process? We provided an integration testing strategy to characterize the functional building blocks of FI applications. In addition the integration testing matrices will assist on testing process management.
(c) How testing can assure standard conformance and automation to ensure extensive testing coverage? We utilized the Elvior TestCast T3 tool to automatize unit testing and to ensure standards adoption. We presented an example case of NGSI9/10 interfaces testing.

The testing starts at the module level and works outward the integration of the cloud application. Based on the empirical analysis the testing techniques presented could be used at different points of the overall cloud enabler engineering process. The cloud application developer could take advantage of the testing activities of the modular parts in order to conduct testing for the larger group and ensure verification (application works as expected) and validation (compliance to the requirements). In addition, we propose the use of an independent tester in order to remove the conflict of interest that is characteristic when the developer tests its own product.

In general the test execution of cloud enablers could be difficult since it involves a number of actors such as (a) the enabler and (b) the use case application developer. Based on the methodology we executed various test cases from the perspective of both actors. Another vital requirement is the correct depiction of the use case application requirements, since these will play an important role to the identification of the test cases. Here, the assumption is that the tester will have access to the functional building blocks of the application and the chains of enablers' dependencies.

The top down and bottom up approaches will warranty that in integration testing (a) the cloud enabler will be tested according to its specification and (b) according to the requirements of the use case application. We encourage its usage since it provides ready-made unit tests for cloud enablers ensuring adoption of standards. An example was the test cases of the FI-STAR Event Service that demonstrated an error regarding the expected output of a test (that was not NGSI9/10 compliance). This test has revealed a hidden error that the SE owner did not identify at the beginning. Finally, the SE owner considered the results and made appropriate corrections to the beta version of the software. It should be mentioned that test cases, test assets, and test features could be accessible could be open to everyone.

The following is a summary of the innovative features of this work.

(a) We focused on the research gap created, regarding cloud enablers testing, in the area of FI services. Thus we present a methodology to allow efficient testing of cloud enablers belonging to different owners, deployed to various cloud platforms and utilized by different cloud applications.
(b) We presented a methodology that incorporates strategies for unit testing (white and black box approaches) and integration testing (bottom up and top down approaches). The correlation of these will ensure an increased coverage of cloud enablers testing.
(c) We presented the FI-STAR and FIWARE projects as use case scenarios of our methodology by including example cases of real world cloud enablers.
(d) We detailed unit and integration testing matrices to allow efficient capturing of testing requirements. In many instances various FI-STAR SE testers with success have utilized the proposed matrices.
(e) We demonstrated the TestCast tool, an advanced solution for testing according to standards such as compliance with ETSI, OMA, TTCN-3, and NGSI9/10. In many instances the tool surfaced failures and issues that did not identified by the white box testing process.

101.6 Conclusions

In this work we presented the Elvior TestCast T3 tool that includes FIWARE compliance. The proposed solution assists cloud enabler testers to rapidly isolate errors and failures based on the virtualization of services so to identify dependencies with other components. Different from other solutions, the approach provides a complete end-to-end transaction that integrates chains of Applications, SEs and GEs. To demonstrate effectiveness, we presented the real world case scenario of FIWARE and FI-STAR projects. We emphasized the adaption of the proposed methodology and tools to the production of test cases; analysis of data and automation of processes based on real world use case requirements for cloud-based healthcare applications. The environment is highly complex with multi-tenancy, diversity of requirements security and off-premises cloud services that are decentralized in terms of ownership. We anticipate that the methodology will serve cloud testers to ensure proper functionality and test coverage and application developers could utilize results as a reference model for building innovating systems.

References

1. Sotiriadis, S., Zampognaro, P., Petrakis, E., & Bessis, N. (2015). Automatic deployment of cloud services for healthcare application development in FI-STAR FP7. *The 30th IEEE international conference on advanced information networking and applications (AINA-2016)*, Le Régent Congress Centre, Crans-Montana, Switzerland, March 23–25, 2016.
2. Seijas, P. L., Li, H., & Thompson, S.. (2013). Towards property-based testing of RESTful web services. *Proceedings of the twelfth ACM SIGPLAN workshop on Erlang* (pp. 77, 78).
3. Incki, K., Ari, I., & Sozer, H.. (2012). A survey of software testing in the cloud. In *Proceedings of the 2012 I.E. sixth international conference on software security and reliability companion (SERE-C '12)*. IEEE Computer Society, Washington, DC (pp. 18–23).
4. Lima Neto, C. R., & Garcia V. C. (2013). Cloud testing framework. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE '13)*. ACM, New York (pp.252, 255).
5. Vakanas, L., Sotiriadis, S., & Petrakis, E. (2015). Implementing the cloud software to data approach for OpenStack environments, adaptive resource management and scheduling for cloud computing, held in conjunction with PODC-2015, Donostia-San Sebastián, Spain, on 20 July 2015.
6. Ciortea, L., Zamfir, C., Bucur, S., Chipounov, V., & Candea, G. (2010). Cloud9: a software testing service. *ACM SIGOPS Operating Systems Review Archive, 43*(4), 5, 10.
7. Banzai, T., Koizumi, H., Kanbayashi, R., Imada, T., Hanawa, T., & Sato, M. (2010). D-Cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology. *Proceedings of 10th IEEE/ACM international conference on cluster, cloud and grid computing, 2010* (pp. 631, 636). Heraklion.
8. Mathew, R., & Spraetz, R. (2009). Test automation on a SaaS platform. *Proceedings of international conference on software testing verification and validation, 2009* (pp. 317, 325). Seoul, Republic of Korea.
9. Wickremasinghe, B, Calheiros, R. N., & Buyya, R. (2010). CloudAnalyst: a CloudSim-based visual modeller for analysing cloud computing environments and applications. *24th IEEE international conference on advanced information networking and applications (AINA), 2010*. Bethlehem.
10. Sotiriadis, S., Bessis, N., Antonopoulos, N.,& Anjum, A. (2013). SimIC: designing a new inter-cloud simulation platform for integrating large-scale resource management. *27th IEEE international conference on advanced information networking and applications (AINA-2013*), March 25–28, Barcelona, IEEE Computer Society, Washington, DC (pp. 90–97).
11. Oriol, M., & Ullah, F. (2010). YETI on the cloud, software testing, verification, and validation workshops (ICSTW). *2010 third international conference on* , vol., no., pp.434, 437, 6–10 April 2010. Madrid.
12. Vilkomir, S. (2012). Cloud Testing: A state-of-the-art review. *Information and Security. An international journal, 28*(2), 213, 222.

13. Grabowski, J., Hogrefe, D., Réthy, G., Schieferdecker, I., Wiles, A., & Willcock, C. (2003). An introduction to the testing and test control notation (TTCN-3). *Computer Networks, 42*(3), 375, 403.
14. Heckel, R., & Lohmann, M. (2005). Towards contract-based testing of web services. *Electronic Notes in Theoretical Computer Science, 116*(2005), 145–156.
15. Wang, J., & Meng, F.. (2011). Software testing based on cloud computing. *In Proceedings of the 2011 international conference on internet computing and information services (ICICIS '11)*. IEEE Computer Society, Washington, DC (pp.176, 178).
16. Sotiriadis, S., Lehmets, A., Petrakis, E., & Bessis, N. (2016). Unit and integration testing of future internet cloud services. *The 31st IEEE international conference on advanced information networking and applications (AINA-2017)*, Tamkang University, Taipei, Taiwan, March 27–29, 2017.