Microservices validation: Mjolnirr platform case study

D.I. Savchenko*, G.I. Radchenko* and O. Taipale**
*South Ural State University, Chelyabinsk, Russia
**Lappeenranta University of Technology, Finland
dmitry.savc@gmail.com, radchenkogi@susu.ac.ru, ossi.taipale@lut.fi

Abstract – Microservice architecture is a cloud application design pattern that implies that the application is divided into a number of small independent services, each of which is responsible for implementing of a certain feature. The need for continuous integration of developed and/or modified microservices in the existing system requires a comprehensive validation of individual microservices and their co-operation as an ensemble with other microservices. In this paper, we would provide an analysis of existing methods of cloud applications testing and identify features that are specific to the microservice architecture. Based on this analysis, we will try to propose a validation methodology of the microservice systems.

Keywords - Microservices, Services Oriented Architecture, Cloud computing, PaaS, testing, validation

I. INTRODUCTION

The microservice architecture is a cloud application design pattern that implies that the application is divided into a number of small independent services, each of which is responsible for implementing of a certain feature. Microservices can be considered as meta-processes in a Meta operating system (OS): they are independent, they can communicate with each other using messages and they can be duplicated, suspended or moved to any computational resource and so on.

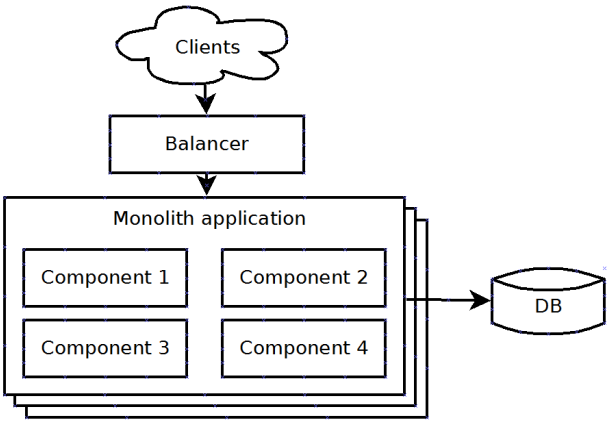

Figure 1. Monolithic system architecture

There are several examples of software systems, which were designed and implemented in accordance with microservice concept. Such companies as Netflix [19] and SoundCloud [3] use the microservice concept in their architectures, while Amazon [20] offers services like S3, which can be considered as microservices.

We can describe microservice architecture by comparing it with the monolithic architecture style, when an application built as a one big single unit (figure 1). The server-side monolithic application will handle HTTP requests, execute domain logic, retrieve and update data from the database, and select and populate HTML views to be sent to the browser. Such a design approach may result in that a change made to a small part of the application, requires the entire monolith to be rebuilt and deployed. Over time, it is often hard to keep a good modular structure, making it harder to keep changes that ought to only affect one module within that system [13].

In a case of microservice architectural style (figure 2), we can divide the application into a several independent components (microservices) and scale them independently. It is useful in a case of high-loaded systems and systems with lots of reusable modules.

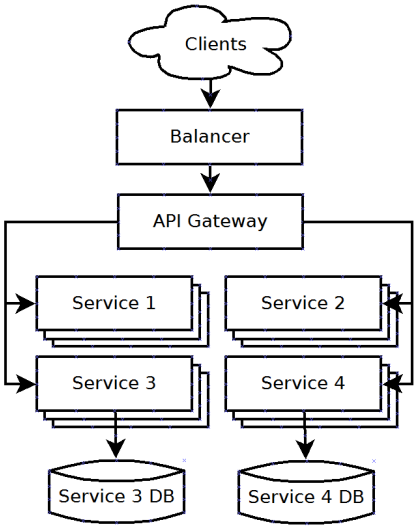

Figure 2. Microservice system architecture

Some developers and researchers believe that the concept of microservices is a specific pattern of implementation of Service Oriented Architecture [11]. In particular, the developers of the Netflix platform, which is considered the embodiment of the microservice architecture, named their approach of a design of software

The reported study was partially supported by RFBR, research project No. 14-07-00420-a and by Grant of the President of the Russian Federation No. MK-7524.2015.9

architecture as "a fine grained Service Oriented Architecture" [12]. However, the microservice pattern defines following specifics of software services development process [13]:

- microservices should use lightweight mechanisms for the communication (often an HTTP resource API);
- microservices should be built around business capabilities;
- microservices should be independently deployable by fully automated deployment machinery;
- there should be a bare minimum of centralized management of these services, architecture should provide decentralized service governance and decentralized data management.

However, the microservice approach is not an ultimate solution to deal with the system complexity. Using microservice approach, we are shifting the accidental complexity [2] from inside of the application out into infrastructure. That is why we should use special approaches to manage that complexity. Programmable infrastructure allows automating the microservices lifecycle management [18]; including automation of service deployment and termination, load balancing, service naming, continuous integration and delivery etc. Such support can be provided by a PaaS cloud platform, in order to scale the system automatically, manage it and provide middleware for the message communication. In addition, there should be implemented monitoring mechanisms to control the flow of the microservice application. It can be implemented as a specialized PaaS [16] solution or integrated in the existing PaaS.

Since the continuous integration is the only option of the implementation of the microservices architecture pattern, the question arises about the support this process on the methodological and software level. One of the most important stages of the process of continuous integration and deployment is validation of the developed software.

Our research goal is to develop the microservice validation methodology and a software support for microservice testing. To reach this goal, during our research we should solve the following tasks:

- to provide a review of existing software validation methods to identify testing features that are specific to microservice architecture;
- to develop a methodology for microservice systems validation, including approaches for component and integration testing;
- to develop a software solution that provides support for microservice systems testing on the basis of the proposed methodology.

In the framework of the current article, we would concentrate on the analysis of the current software validation standards and approaches and their relevance to the microservice architecture pattern. This paper is organized as follows. In Section II we will provide a review of the research and standards, related to the microservices testing and validation. In Section III we would try to identify special aspects of microservice validation, related to the component, integration and system testing. In Section IV we would propose microservice validation techniques for the Mjolnirr cloud platform prototype. In section V we summarize the results of our research and further research directions

II. Related research

Microservices concept is relatively new, so there are few materials in the field of microservice validation and testing right now. Basic information about the microservice approach is provided in the article [13] by James Lewis and Martin Fowler. In this article, they define the concept of microservice as well as the specific features of microservice architecture. Toby Clemson in his article [5] provides strong emphasis on the analysis of possible strategies of microservice systems testing, including certain types of microservice systems architecture; unit, integration, component and contract testing. Neal Ford provides a review of the proposed approaches on this subject is his work [6]. He describes basic ideas about microservice systems development, monitoring and testing. In addition, several commercial companies successfully moved to microservice architectures and published their result. For example in the paper [14] Netflix provide an overview of their PaaS system and mechanisms that they use to provide features of their platform to the components that initially was not designed to be used inside of the Netflix PaaS platform.

In our previous works, we described a prototype of distributed computing system called Mjolnirr [17]. This system works with isolated components, connected using message passing pattern. This system also can be considered as a microservice platform, because each component is isolated, fine-grained and has open standardized interface. In addition, automated scheduling and planning mechanisms have been implemented within the Mjolnirr platform in order to optimize the system load [15].

The ISO/IEC standards, ISO/IEC 25010 – Software Product Quality [7] and ISO/IEC 29119 – Software Testing [8], serve as the starting point of the study of microservice system testing and validation. Generally, we define testing as a combination of verification and validation [8]. In this research, we define microservice testing as microservice validation, because in general it is impossible to apply static testing methods to the compiled microservice.

As we mentioned before, we can consider microservices as a subset of Service Oriented Architecture (SOA). Thus, we can use SOA validation techniques in application to microservices systems. Some of existing approaches to SOA validation use SOA's BPEL business model for constraints-based SOA functional testing [10], other use white-box testing [1] to determine, is service covered with tests enough.

When we are talking about service testing and validation, we should decide, which quality characteristics are important for the product we are going to validate. Quality in use is defined in the ISO/IEC 25011 standard: “Quality in use of a service is the degree to which a

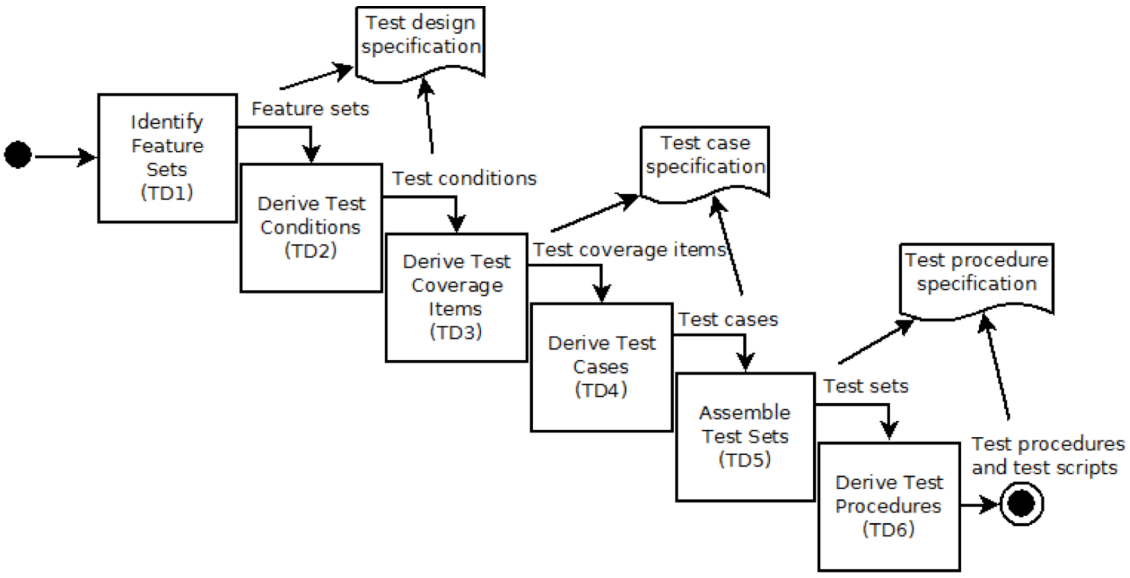

Figure 3. ISO/IEC/IEEE29119-2 Test Design and Implementation Process (modified)

service can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, satisfaction, freedom from risk and SLA coverage" [8]. ISO/IEC 25011 standard defines eight quality characteristics of services: Service Suitability, Service Usability, Service Security, Service Reliability, Service Tangibility, Service Responsiveness, Service Adaptability, and Service Maintainability. All of them, except Service Tangibility, can be applied to microservices. In addition, in a case of microservice system, we do not care about effectiveness and efficiency – these characteristic should be provided and measured by application programmer. However, satisfaction (stability of the whole system, quick response, etc.), freedom from risk and SLA coverage are important when we want to validate microservice system.

According to ISO/IEC 29119-4 standard, test design and implementation process consists of 6 steps – identifying feature sets (TD1), deriving test conditions (TD2), deriving test coverage items (TD3), deriving test cases (TD4), assembling test sets (TD5) and deriving test procedures (TD6) (see Fig. 3) [8]. To describe the microservice validation methodology, we should focus on the TD2, TD3, TD4 and TD6 steps.

III. Special Aspects of Microservice Validation

In this study, we define microservice as a software-oriented entity, which has following features:

- *Isolation* from other microservices as well as from the execution environment based on a virtualized container;
- *Autonomy* – microservices can be deployed, destroyed, moved or duplicated independently. Thus, microservice cannot be bound to any local resource because microservice environment can create more than one instance of the same microservice;
- *Open and standardized interface* that describes all available communication methods (either API or GUI);
- *Microservice is fine-grained* – each microservice should handle its own task.

Let us consider the most significant stages of the validation process, in relation both to the software as a whole and in particular microservice systems and define the features of the validation to be considered for microservice systems. By *unit validation* we understand individual validation of the microservice, by *integration validation* – validation of the interaction between individual microservices and by *system validation* – validation of the whole system (see Table 4).

System validation does not seem to offer features to microservices testing, since validation of the system level does not consider the internal structure of the application. The internal structure of the system can be considered as a black box, so there would be no difference in the system validation process for any application for which network resources access is needed.

A. Microservice unit validation

Functional Unit validation of microservice systems involves independent validation of individual microservices to meet the functional requirements. This

TABLE I. Featured Types and Levels of Testing

	Component	Integration	System
Functional	+	+	–
Load	+	+	–
Security	+	+	–
– - no features for microservice validation + - has features for microservice validation			

type of validation is based on a formal description of the functional requirements (including requirements as input and output data) imposed on each microservice. To provide the functional unit validation, microservice does not require to be deployed in a cloud environment, as this type of validation considers microservice as an isolated component.

Development of a software system in accordance with microservice approach is a process of development of individual independent microservices interacting exclusively based on open protocols. Thus, each microservice represents independent software product that should be developed and tested independently of the other microservices. Functional unit validation of the microservices can be divided into two stages (see Fig. 7):

1. unit, integration and system validation of the microservice source code or/and internal components (basic level);
2. self-validation of the microservice interface.

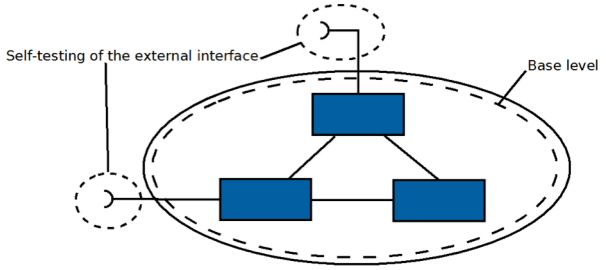

Figure 4. Functional unit validation of the microservice

Each microservice can be a relatively complex software system, which consists of several software components (local storage, web server, etc.) encapsulated in a container. Even in a case of ready-made software solutions, integration of such a component ensemble inside a container must be accompanied by validation the correctness of these components work together.

After that, it is necessary to test the interface provided by the individual microservice on its compliance with the specifications defined in the design. In this case, the entire microservice should be tested in general using its external interface.

Functional unit tests of a microservice should be conducted automatically on each microservice configuration change.

Load unit validation is aimed at the individual microservice validation at a certain load. Such validation may also be performed automatically for each microservice. Microservice load validation allows to identify resources that should be provided to the microservice container to ensure its correct work.

Security unit validation is aimed at the security and isolation validation of the individual microservice. As in the case of functional validation, microservice security does not end with the security of its individual classes. In component security validation we test the security of the whole microservice in the complex (for example, for typical vulnerabilities of web servers, injections, etc.). This stage of validation is performed without the presence of application programmer.

B. *Integration microservice validation*

Integration validation of microservice systems involves validation of all types of communications between the microservices. It includes validation of communication protocols and formats, resolution of deadlocks, shared resource usage and messaging sequence. To provide all of this validation techniques, we need to track all the messages, transmitted between microservices, and build the messaging graph. For this purpose, we need to know the messaging direction, message origin and destination. This information must be provided by microservice standard contract.

Functional integration validation ensures interoperability of the validation individual microservices. In this type of validation, we check for correctness of sending requests from one microservice to another, the sequence of interactions, microservice orchestration and choreography. In this type of validation, we can generate test cases in semi-automated mode – we need to know the communication sequence, and this data should be provided by the developer. Using this data, we can produce communication graph and test the system according to this graph.

Load integration validation is aimed at checking the microservice correctness under automatic deployment, backup and transfer, as well as microservice orchestration and choreography. This type of validation can be done automatically. Unlike functional integration validation, this type of validation includes loading of communication channels to discover the maximum load the microservice can handle before failure.

Integration security validation is aimed at validation of the security of communications between microservices, as well as checking for the interception of messages from outside or inside by anyone, besides the message recipient. Microservice specific validation should also take into account the particular environment in which the microservices are applied.

IV. VALIDATION OF MJOLNIRR-BASED MICROSERVICES

We have developed a prototype of the Mjolnirr microservice platform [17]. This platform supports microservices, implemented using JVM-compatible languages and Java-based containerization. Integration with Mjolnirr will be the first step of our validation framework implementation. This step includes Java-specific methods of software validation, such as jUnit for component validation. Let's describe, how validation implementation steps that we have highlighted in Chapter 2 could be implemented on the basis of Mjolinrr platform.

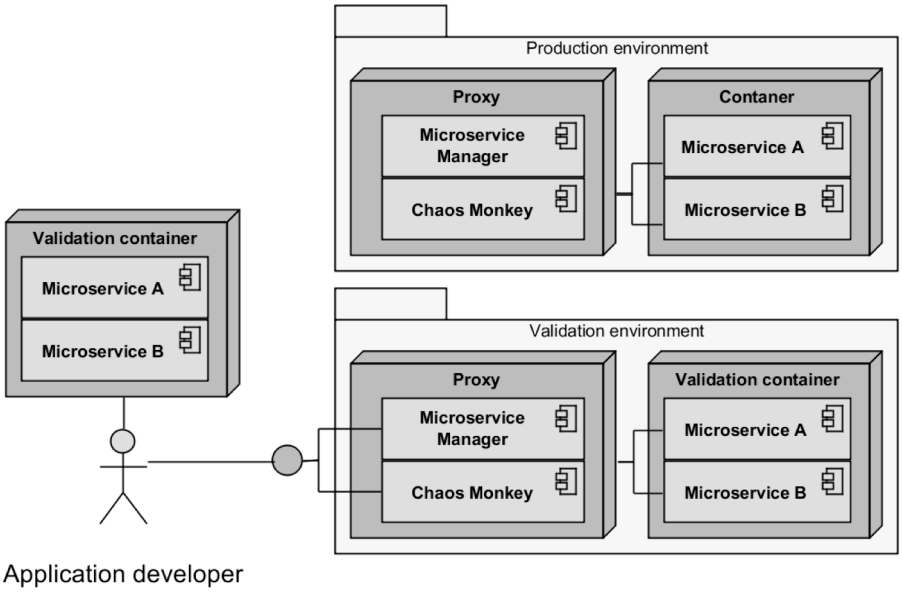

Figure 6. Validation of Mjolnirr-based microservices

TD2 (Derive test conditions) describes the testable aspects of components, in our case - microservices. In Mjolnirr, we can consider microservices and their calls as test conditions. As mentioned in Chapter 3, each microservice is isolated and autonomous and can be tested in a fully isolated JVM. Microservices are communicating using messages. Therefore, conditions, which determine quality of microservice system, are following: full isolation for component level, message communication for integration level and encapsulation for system level.

TD3 (Derive test coverage items) describes the attributes, derived from test conditions. Test conditions are microservices and their communications, therefore, test coverage items would be microservice methods and messages. Each microservice method and communication channel should be covered. Coverage items can be covered automatically or by application developer.

TD4 (Derive test cases) describes the test cases generation process. According to quality characteristics and ISO/IEC 29119-4 standard, we can decide, what test design techniques we should use and find exact technologies to implement those techniques. To perform a component-level testing, we can use only specification-based techniques, such as Boundary Value Analysis, Equivalence partitioning, Use case testing, Random testing and Scenario testing. In a case of component security testing, we can also scan microservice for typical vulnerabilities of the specific middleware, used in its infrastructure.

TD6 (Derive test procedures) describes the testing process. In Mjolnirr we use both automated and user-provided test cases, so we should cover all the coverage items, provided by microservice interface, and call all the test cases, provided by application developer. Test case can modify the microservice environment and state, therefore, microservice validation should be executed in an isolated environment, which should be purged after each set execution. As shown on figure 6, Mjolnirr has three sets of environments – special validation container for isolated component-level validation, validation environment for integration-level validation and production environment for end users. Validation environment and production environment has built-in Chaos Monkey service to ensure system stability. These three steps should be passed by each microservice to go to production.

To use those techniques, we need to have detailed microservice interface description, including input and output variables types, boundaries and syntax. Using this information, we can perform black box testing to each microservice in addition to the simple unit tests. Microservice interface definition described on figure 7.

```
{
  "name": "test_microservice",
  "inputs": [
    {
      "type": "integer",
      "min": 0,
      "max": 10
    },
    {
      "type": "string",
      "syntax": "he.?lo"
    }
  ],
  "outputs": [
    {
      "type": "integer",
      "min": 0,
      "max": 10
    }
  ],
  "input_connections": ["test_microservice2"],
  "output_connections": ["test_microservice3"]
}
```

Figure 7. Microservice interface description

In a case of integration testing we have knowledge about internal structure of the tested system. Therefore, we can use the testing techniques, mentioned before, and structure-based test design techniques, such as data flow testing, branch condition testing, etc. As shown on figure

5, microservice interface also describes input and output connections for each microservice, and we can use this knowledge to produce call graph and validate this structure.

Integration-level validation cannot be implemented on a container level, so this type of validation will be handled by Proxy node. Proxy will store the call graph, compare it with the desired structure, based on component interface. Also, Proxy will use different integration validation techniques, like Chaos Monkey [4] and random messaging, which are useful in a case of high load system.

V. CONLUSION

In this article, we described microservice testing features and validation techniques. In addition, specific test design techniques were mentioned. We have described the basics of microservice approach, provided and overview of methods and standards of software testing and validation. We described microservice validation features and proposed an approach to microservice validation. At last we proposed a brief overview of possible methods of implementation of microservice validation framework based on a Mjolnirr platform.

As a further development of this study, we will implement microservice validation framework, which will be able to perform automated and semi-automated validation to the microservices.

REFERENCES

[1] Bartolini C., Bertolino A., Elbaum S., Marchetti E. Whitening SOA testing. *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering.* 161—170.

[2] Brooks F.P. No Silver Bullet. Essence and accidents of software engineering. IEEE computer 20(4); 1987: 10-19.

[3] Calchado P. Building Products at SoundCloud—Part III: Microservices in Scala and Finagle. [Online] Available at: https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-3-microservices-in-scala-and-finagle [accessed 6.02.2015].

[4] Chaos Monkey. [Online] Available at: https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey [accessed 12.01.2015]

[5] Clemson, T. Testing Strategies in a Microservice Architecture. 2014. [Online] Available at: http://martinfowler.com/articles/microservice-testing/ [Accessed 10.01.2015].

[6] Ford N. Building Microservice Architectures. [Online] Available from: http://nealford.com/downloads/Building_Microservice_Architectures(Neal_Ford).pdf [accessed 13.02.2015]

[7] International Organization for Standardization (2014) 25010. Service Quality Requirements and Evaluation (SquaRE) – System and software Quality Model, 2014, Canada.

[8] International Organization for Standardization (2014) 29119. Software Testing, 2014, Canada.

[9] International Organization for Standardization (2015) 25011. Service Quality Requirements and Evaluation (SquaRE) – Service Quality Model, 2015, China.

[10] Jehan S., Pill I., Wotawa F. Functional SOA Testing Based on Constraints. *Proceedings of the 8th International Workshop on Automation of Software Test.* 33—39.

[11] Jones S. Microservices is SOA, for those who know what SOA is, 2014. [Online]. Available at: http://service-architecture.blogspot.ru/2014/03/microservices-is-soa-for-those-who-know.html. [Accessed: 17.11.2014].

[12] Kant N., Tonse T. Karyon: The nucleus of a Composable Web Service. [Online] Available at: http://techblog.netflix.com/2013/03/karyon-nucleus-of-composable-web-service.html [accessed 17.11.2014].

[13] Lewis J., Fowler M. Microservices. [Online] Available at: http://martinfowler.com/articles/microservices.html [accessed 17.11.2014].

[14] Prana: A Sidecar for your Netflix PaaS based Applications and Services. [Online] Available from: http://techblog.netflix.com/2014/11/prana-sidecar-for-your-netflix-paas.html [accessed 13.02.2015]

[15] Radchenko G., Mikhailov P., Savchenko D., Shamakina A., Sokolinsky L. Component-based development of cloud applications: a case study of the Mjolnirr platform. *Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR '14).* Article 6, 10 pages.

[16] Rhoton J, Haukioja R. Cloud computing architected. [Tunbridge Wells, Kent]: Recursive Press; 2011.

[17] Savchenko D., Radchenko G. Mjolnirr: A Hybrid Approach to Distributed Computing Architecture and Implementation // CLOSER 2014. Proceedings of the 4th International Conference on Cloud Computing and Services Science (Barcelona, Spain 3-5 April, 2014), 2014. P. 445-450.

[18] Thones J. Microservices. Software, IEEE, Volume 24 Issue 3, 2015. P. 116-116.

[19] Tonse S. Microservices at Netflix. [Online] Available from: http://www.slideshare.net/stonse/microservices-at-netflix [accessed 6.02.2015].

[20] Varia J. Cloud architectures. White Paper of Amazon Web Services, 2008. [Online] Available at: https://media.amazonwebservices.com/AWS_Cloud_Architectures.pdf: 16. [accessed 17.11.2014].