Generic Refactoring Methodology for Cloud Migration
Position Paper

Manoj Kesavulu, Marija Bezbradica and Markus Helfert
Lero – Irish Software Research Organization,
School of Computing, Dublin City University, Dublin, Ireland

Keywords: Generic Architectural Refactoring, Cloud Migration, Service-oriented Architecture, Cloud Platform.

Abstract: Cloud migration has attracted a lot of attention in both industry and academia due to the on-demand, high availability, dynamic scalable nature. Organizations choose to move their on-premise applications to adapt to the virtualized environment of the cloud where the services are accessed remotely over the internet. These applications need to be re-engineered to completely exploit the cloud infrastructure such as performance and scalability improvements over the on-premise infrastructure. This paper proposes a re-engineering approach called *architectural refactoring* for restructuring on-premise application components to adopt to the cloud environment with the aim of achieving significant increase in non-functional quality attributes such as performance, scalability and maintainability of the cloud architectures. This paper proposes, when needed to migrate to cloud, the application is divided into smaller components, converted into services and deployed to cloud. The paper discusses existing issues faced by software developers and engineers during cloud migration, introduces architectural refactoring as a solution and explains the generic refactoring process at an architectural level.

1 INTRODUCTION

Cloud computing has become a buzzword in the IT industry today and it is regarded as the most influential technology in the present time. Given the long term benefits of adopting to this promising technology, many industries decided to migrate their on-premise applications to cloud. Cloud migration is defined a process of partially or completely moving an organization's digital assets, services, IT resources or applications to a cloud platform with an intent to significantly improve various quality attributes such as performance, scalability, maintainability and similar (Pahl et al. 2013). Moving an application using "lift-and-shift" approach, (which is moving it as it is), moves all the existing issues associated with it to cloud and it is likely to introduce new concerns and challenges. An alternative approach to move legacy on-premise application to cloud is "Big Bang" rewrite, (where the application is built in cloud from scratch), is not only a time-consuming process but is also extremely risky and will likely end in failure. Another recent option is to follow service-oriented approach where the target application is divided into smaller components and converted into services comprising of core business functions, before its migration to the cloud. this also involves making sure that new features are exploited and the entire application will continue to working in the new environment (Garg et al. 2016).

Migrating an application to a cloud platform is difficult, costly and error-prone (Kwon & Tilevich 2014). A major part of the IT system are applications which are integrated and support core business process and services, many of which are used for utility needs, and these are non-core applications, meaning that components of these applications are loosely coupled. These application components need to be re-engineered before deploying to cloud for better use of the services offered by the cloud platform. There are scenarios where an application is deployed over more than one cloud platform which is termed as multi-cloud deployment (Jamshidi et al. 2015).

This paper presents a generic architectural refactoring approach that facilitates the process of transforming on-premise applications to use cloud-based services. The paper is structured as follows: Section 2 gives an overview of the cloud platform. Section 3 discuss cloud migration and identifies

Kesavulu, M., Bezbradica, M. and Helfert, M.
Generic Refactoring Methodology for Cloud Migration - Position Paper.
DOI: 10.5220/0006373106920695
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 664-667
ISBN: 978-989-758-243-1

issues at an architectural level. Section 4 introduces a generic refactoring methodology and at last Section 5 describes conclusions and future work.

2 CLOUD PLATFORM

Traditional IT (Information Technology) aligns resources according to the way the applications are deployed within dedicated infrastructure and data storage to fulfil business requirements. Cloud computing (CC) has emerged as a computing paradigm with benefits such as high scalability, reduced IT costs, on demand self-service, pay-as-you-go price models, elasticity in provision computing resources. It achieves this by varying workload so organisations became attracted to move their on-premise systems to the cloud.

CC can be mainly divided into three service layers as shown in Figure 1 below:

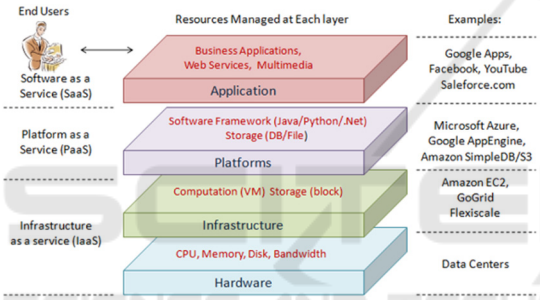

Figure 1: Cloud Service Layers (Petrolo et al. 2014).

- *Infrastructure as a Service (IaaS)* offers computing resources, both physical and virtual, for processing and storage.
- *Platform as a Service (PaaS)* offers development environment for software developers to write their applications on a particular platform without worrying about the underlying hardware infrastructure.
- *Software as a Service (SaaS)* offers software applications that can be accessed and used by the end-users.

In addition to the layers, some other layers are introduced namely Data as a Service, Everything as a Service, Network as a Service, Things as a Service and Sensing and Actuation as a Service (SAaaS) and so on. CC has five deployment models (1) Private cloud (2) Public Cloud (3) Hybrid Cloud (4) Community Cloud (Marinos & Briscoe 2009) and (5) Multi-Cloud (Paraiso et al. 2012). With the the abundance of options to be considered for the selection of a suitable cloud service and deployment model, organization has to decide on the following aspects (Garg et al. 2016):

- Selection of Deployment Model
- Selection of Service model
- Selection of Appropriate Service Package

Upon making a suitable decision, the target application should be re-engineered to fit the selected cloud platform adhering to the chosen deployment and service model. An important pre-requisite to be fulfilled, while transforming application components serving the underlying business functions, is that they preserve the external behaviour after the migration process. According to a survey conducted by Kratzke & Quint (2017), cloud specific design methodologies are yet to be designed and developed. This paper introduces an early stage generic refactoring methodology which guarantees successful application of architectural refactoring while migrating on-premise applications to cloud. In the next section, we discuss cloud migration process in more details.

3 CLOUD MIGRATION

Cloud migration is the process of moving applications, services, code, and business logic deployed on the on-premise infrastructure to cloud platforms with the aim of achieving significant improvement in performance, scalability and cost reduction. There are various methodologies and approaches discussed in literature regarding cloud migration. Rowe, Brinkley and Tabrizi (2013) designed a generic methodology for the migration of legacy system to Cloud Platform describing the following steps: (1) Architectural representation of the legacy application; (2) Redesign the architecture model; (3) MDA transformation; (4) Web service generation; (5) Web service-based invocation of legacy functionalities; (6) Selection of suitable Cloud Computing Platform; (7) Web service deployment in the service cloud. Considering the steps (2) and (3), where the original architecture model is redesigned to identify services that can be provided in a SaaS architecture as shown in Figure 2 below:

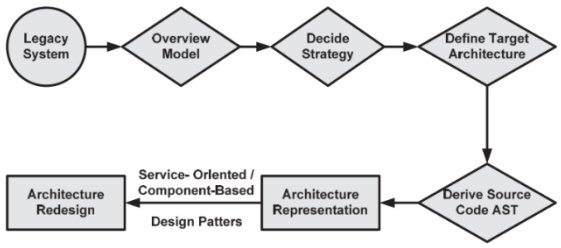

Figure 2: Refactoring with Architecture Driven Modernization (Rowe et al. 2013).

This methodology explains the prerequisite steps followed in the refactoring process rather than the actual refactoring process.

The authors Kwon and Tilevich (2014) describe an automated transitioning approach to cloud-based services and discuss three scenarios: moving a part of application functionality to cloud, adding new (fault-tolerance) functionality and switching an application to use an alternate cloud-based service. These three scenarios use cloud refactoring approaches as a solution. However, in a real-world situation these approaches can only be applied to the scenarios discussed or similar scenarios and do not serve as a generic refactoring approach which could be applied to all cloud migration scenarios. Hence there is a crucial need to derive a generic refactoring process.

4 GENERIC REFACTORING METHODOLOGY

Refactoring is a process of changing internal design of the system while preserving its external behaviour (Fowler 2002). The goal of refactoring is to improve a certain quality while preserving others. It is a bottom-up process which helps clean-up inconsistent or insufficient design decisions which can be applied for design artifacts, models, documents, UML diagrams, processes and architectures (Stal 2007). There have been numerous examples where refactoring is applied in different cases and scenarios with migrating legacy applications to cloud platforms (Zimmermann 2016; Kwon & Tilevich 2014; Rowe et al. 2013; Chauhan & Babar 2012; Schmidt et al. 2012). Refactoring can be simple – move, add, rename, remove, pullup, substitute and so on and complex – combination of two or more simple refactorings.

The process of refactoring can be described as follows (Figure 3):

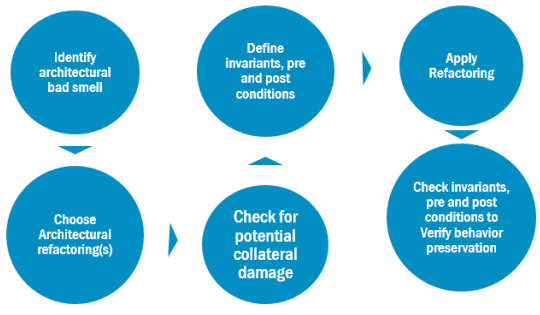

Figure 3: Generic refactoring steps (Kesavulu, et al., 2016).

- *Identify architectural bad smell*: an architectural bad smell is commonly (although not always intentionally) used set of architectural design decisions that negatively impacts system quality (Stal 2007). Another definition for architectural bad smell is the observation or the suspect that something in architecture design and its implementation is no longer adequate (i.e., good enough) under the actual requirements and current constraints for the system (Zimmermann 2016). Some of the examples are: high coupling between subsystems or a data store which cannot handle concurrent queries in reasonable time(Zimmermann 2016); the former example is a general architectural bad smell whereas the latter is a cloud specific bad smell. Though there are cloud specific bad smells and corresponding architectural refactoring is available in the form of catalogues, cloud specific techniques for identification of smells and tools for application of refactoring are deficient. Our future work includes exploration of cloud specific bad smell detection techniques.
- *Choose Architectural Refactoring(s)*: Choosing an appropriate architectural refactoring or a set of refactorings to apply. Almost every bad smell is associated with a potential architectural refactoring. But there are various aspects to be considered while choosing a refactoring such as each refactoring should be applied incrementally in small steps and specificity to cloud environments.
- *Check for potential collateral damage*: A refactoring may trigger other design transformations due to existing dependencies between application (or service) components, some of which may be dangerous.
- *Define Invariants, Pre and Post Conditions*: Before applying any refactoring, define necessary steps to be taken to verify the preservation of behavior of the system after applying the refactoring.
- *Apply Refactoring*: The suitable target refactoring chosen in step 2 is applied to the legacy system (application) architecture incrementally. Some of the refactorings are Virtualize Server for IaaS layer, Swap Messaging Provider for PaaS layer of the cloud and so on.
- *Check Invariants, Pre and Post Conditions*: The Invariants, Pre and Post conditions defined in step 3 are verified which proves that the behavior of the target system is preserved and the refactoring is successfully applied.

This process can be considered as a generic architectural refactoring methodology which can be

used to verify the successful application of refactoring to any cloud migration scenario.

5 CONCLUSIONS AND FUTURE WORK

This paper introduces the cloud platform, discuss its advantages over traditional IT platform and the need for cloud migration and issues identified. The paper proposes a generic architectural refactoring methodology and introduces its steps while explaining in detail. This methodology can be used to verify successful application of refactoring process in various scenarios of cloud migration.

The future work to this project would be to develop a concept of detailed refactoring techniques which includes methods to identify architecture smells including exploration of existing architectural smells identification and behaviour preservation techniques; tools to recommend and apply architectural refactoring specific to cloud service architectures using a case study or a real-world example. Evaluation of the methodology will be conducted after choosing a suitable cloud environment.

ACKNOWLEDGEMENTS

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

REFERENCES

Chauhan, M.A. & Babar, M.A., 2012. Towards Process Support for Migrating Applications to Cloud Computing. *2012 International Conference on Cloud Computing and Service Computing (Csc)*, pp.80–87.

Fowler, M., 2002. Refactoring: Improving the Design of Existing Code. In D. Wells & L. Williams, eds. *Extreme Programming and Agile Methods --- XP/Agile Universe 2002: Second XP Universe and First Agile Universe Conference Chicago, IL, USA, August 4--7, 2002 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 256. Available at: http://dx.doi.org/10.1007/3-540-45672-4_31.

Garg, R., Heimgartner, M. & Stiller, B., 2016. Decision support system for adoption of cloud-based services. *CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science*, 1(Closer), pp.71–82. Available at: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84979743405&partnerID=40&md5=9ae740659dbd9271f229e7cc1feaaf05.

Jamshidi, P. et al., 2015. Cloud Migration Patterns: A Multi-cloud Service Architecture Perspective. In pp. 6–19. Available at: http://link.springer.com/10.1007/978-3-319-22885-3.

Kesavulu, M., Helfert, M. & Bezbradica, M., 2016. Towards Refactoring in Cloud-Centric Internet of Things for Smart Cities. Dublin, s.n.

Kratzke, N. & Quint, P.-C., 2017. Understanding Cloud-native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study. *Journal of Systems and Software*, (January).

Kwon, Y.W. & Tilevich, E., 2014. Cloud refactoring: Automated transitioning to cloud-based services. *Automated Software Engineering*, 21(3), pp.345–372.

Marinos, A. & Briscoe, G., 2009. Community cloud computing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5931 LNCS, pp.472–484.

Pahl, C., Xiong, H. & Walshe, R., 2013. A comparison of on-premise to cloud migration approaches. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8135 LNCS, pp.212–226.

Paraiso, F. et al., 2012. A federated multi-cloud PaaS infrastructure. *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, pp.392–399.

Petrolo, R., Loscrí, V. & Mitton, N., 2014. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities - WiMobCity '14*, 25(3), pp.61–66. Available at: http://dl.acm.org/citation.cfm?id=2633661.2633667.

Rowe, F., Brinkley, J. & Tabrizi, N., 2013. Migrating Legacy Applications to the Cloud. *2013 International Conference on Cloud Computing and Big Data (Cloudcom-Asia)*, (October 2009), pp.68–77.

Schmidt, F., MacDonell, S.G. & Connor, A.M., 2012. An automatic architecture reconstruction and refactoring framework. *Studies in Computational Intelligence*, 377, pp.95–111.

Stal, M., 2007. Refactoring Software Architectures. In *Agile Software Architecture*. Elsevier, pp. 63–82. Available at: http://linkinghub.elsevier.com/retrieve/pii/B9780124077720000034.

Zimmermann, O., 2016. Architectural Refactoring for the Cloud : a Decision-Centric View on Cloud Migration. *Computing*.