Cloud–Computing: from Revolution to Evolution

Sébastien Mosser[⋆], Eirik Brandtzæg[⋆,†], Parastoo Mohagheghi[⋆]

[⋆] SINTEF IKT
[†] University of Oslo
`{firstname.lastname}@sintef.no`

Extended Abstract, submitted to BENEVOL'11

Introduction. Cloud–Computing [1] was considered as a *revolution*. Taking its root in distributed systems design, this paradigm advocates the share of distributed computing resources designated as "*the cloud*". The main advantage of using a cloud-based infrastructure is the associated scalability property (called *elasticity*). Since a cloud works on a *pay–as–you–go* basis, companies can rent computing resources in an elastic way. A typical example is to temporary increase the server–side capacity of an e–commerce website to avoid service breakdowns during a load peak (*e.g.*, Christmas period). However, there is still a huge gap between the commercial point of view and the technical reality that one has to face in front of "*the cloud*". As any emerging paradigm, and despite all its intrinsic advantages, Cloud–Computing still relies on fuzzy definitions[1] and lots of buzzwords (*e.g.*, the overused "*IaaS*", "*PaaS*" and "*SaaS*" acronyms that does not come with accepted definitions).

Problem Statement. A company that wants to migrate its own systems to the cloud (*i.e.*, be part of the cloud *revolution*) has to cope with existing standards. They focus on cloud modeling, but does not provide any support for software evolution. Thus, the *evolution* of a legacy system into a cloud–based system is a difficult task. On the one hand, an immediate issue is the paradigm shift (*e.g.*, migrating a centralized COBOL system to a service–oriented architecture deployed in the cloud). This issue is not directly related to Cloud–Computing, and considered as out of the scope of this work (*e.g*, see the SMART method [2]). On the other hand, the Cloud–Computing paradigm introduces several key concepts that must be used during the evolution process to accurately promote a given legacy system into a cloud–based one. For example, deploying an application to the cloud does not automatically transform it into a scalable entity: the evolution process must carefully identify the components that can be replicated to ensure elasticity using resources available in the cloud. Consequently, the evolution of a legacy system into a cloud–based system requires a dedicated entity that support *(i)* reasoning mechanisms dedicated to

[1]The Cloud–Standard initiative (`http://cloud-standards.org/`) lists dozens of overlapping standards related to Cloud–Computing. They focus on infrastructure modeling or business modeling.

cloud concepts and *(ii)* technical implementation of cloud deployment for such systems.

Objectives. Instead of designing yet another cloud standard, our goal (part of the REMICS project [3]) is to define a language that supports evolution to the cloud. The objectives of this language are the following:

- *Platform–independence.* It is an abstract modeling language to support the description of the software that will be deployed on the cloud. This architecture description language includes cloud–specific concepts (*e.g.*, elastic capability, deployment geographic zone, failure recovery, parallelism, data protection). The language acts as an intermediary pivot between legacy applications and the cloud. Thus, the evolution process does not target concrete cloud providers entities anymore. Moreover, it is possible to reason on the modeled element using a cloud–based vocabulary.
- *Transparent projection.* Based on the modeled entities, the framework associated to the language handle the projection of the abstract description into concrete cloud entities. A matching step is used to accurately bind the abstract resource described in the language with available resources published by cloud providers. For example, at the infrastructure level, it identifies which virtual images must be deployed on what cloud resources.
- *Automated deployment.* The language comes with an interpreter that implements the actual deployment. It abstracts the underlying platform complexity and programming interfaces. Thus, assuming that the evolution process targets the modeling language previously described, the application can be deployed on existing clouds in an automatic way.

Perspectives: Cloud Evolution. The definition of this language and its associated engine is a first step. Based on this experience, we will consider evolution in cloud context according to two complementary axis: *(i)* the evolution of the application after its initial deployment and *(ii)* the evolution of the cloud infrastructure itself.

Acknowledgments. This work is funded by the EU Commission through the REMICS project (`http://remics.eu`), contract number 257793 (FP7).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[2] G. Lewis, E. Morris, D. Smith, and L. O'Brien. Service-Oriented Migration and Reuse Technique (SMART). In *Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice*, pages 222–229, Washington, DC, USA, 2005. IEEE Computer Society.

[3] P. Mohagheghi and T. Sæther. Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project. In *SERVICES*, pages 507–514. IEEE Computer Society, 2011.