This is the draft version of a short paper accepted in the new ideas and work-in-progress section of SEFM 2017 - 15th International Conference on Software Engineering and Formal Methods - September 4-8, 2017 in Trento, Italy

Towards automated deployment of self-adaptive applications on hybrid clouds

Lom Messan Hillah[1,2], Rodrigo Assad[3], Antonia Bertolino[4], Marcio Delamaro[5] Fabio De Rosa[6], Vinicius Garcia[7], Francesca Lonetti[4], Ariele-Paolo Maesano[6], Libero Maesano[6], Eda Marchetti[4], Breno Miranda[7], Auri Vincenzi[8], and Juliano Iyoda[7]

[1] Univ. Paris Nanterre, F-92000 Nanterre, France
[2] Sorbonne Universités, UPMC, CNRS, LIP6 UMR7606, F-75005 Paris, France
lom-messan.hillah@lip6.fr
[3] Ustore
assad@usto.re
[4] ISTI-CNR, 56124 Pisa, Italy,
{antonia.bertolino, francesca.lonetti, eda.marchetti}@isti.cnr.it
[5] Universidade de São Paulo
delamaro@icmc.usp.br
[6] Simple Engineering, F-75011 Paris, France
{libero.maesano, ariele.maesano, fabio.de-rosa}@simple-eng.com
[7] Universidade Federal de Pernambuco
{jmi,vcg,bafm}@cin.ufpe.br
[8] Universidade Federal de São Carlos
auri@dc.ufscar.br

Abstract. Cloud computing promises high dynamism, flexibility, and elasticity of applications at lower infrastructure costs. However, resource management, portability, and interoperability remain a challenge for cloud application users, since the current major cloud application providers have not converged to a standard interface, and the deployment supporting tools are highly heterogeneous. Besides, by their very nature, cloud applications bring serious traceability, security and privacy issues. This position paper describes a research thread on an extensible Domain Specific Language (DSL), a platform for the automated deployment, and a generic architecture of an ops application manager for self-adaptive distributed applications on hybrid cloud infrastructures. The idea is to overcome the cited limitations by empowering the cloud applications with self-configuration, self-healing, and self-protection capabilities. Such autonomous governance can be achieved by letting cloud users define their policies concerning security, data protection, dependability and functional compliance behavior using the proposed DSL. Real world trials in different application domains are discussed.

1 Introduction

Cloud computing is used to provision the physical resources (servers, storage, network) of the digital ecosystem, allowing a substantial optimization of the

operating costs. However, cloud computing is more than a cost-optimizing technology. It bears to users significant features - virtualization, job scheduling, and programmability - allowing the sustainable implementation of robust scalability, availability, and serviceability requirements on commodity hardware.

However, the most notable features of cloud computing, such as virtualization, come at the price of increased security and data protection risks. Moving to a virtualized environment does not free from the security risks already faced in the physical environment, but rather introduces new ones, related to virtual machine and network management, resource exhaustion, hypervisor vulnerabilities, multi-tenancy handling, and cloud access control.

DevOps is an emerging paradigm of integration of the development process within the production stage. To adopt DevOps effectively, cloud application developers have yet to find solutions to tough problems: i) how to design, develop, deploy, and operate efficiently applications that fulfill, on one side, stringent scalability, availability and serviceability needs, and, on the other side, strict security and data protection obligations; ii) how to cope with the heterogeneity and lack of interoperability of cloud infrastructures and the consequent lack of portability of cloud applications; iii) how to combine the advantages of agile, flexible, and continuous integration, testing, delivery, and deployment, and of mission-critical quality assurance, test, and verification.

The main solution of the aforementioned problems is the automation of DevOps jobs, in particular test, configuration, deployment, and ops management. This position paper introduces an ongoing research on automating installation, configuration, startup, and operation management of self-adaptive distributed applications on hybrid cloud infrastructures. In particular, it presents an envisaged solution based on three correlated research topics: (i) a declarative, cloud agnostic, and extensible Domain Specific Language (DSL) for structural and behavioral modeling and policy definition for automated deployment and self-management; (ii) a generic and instantiated architecture of an autonomic ops application manager enabling self-configuration, self-healing, and self-protection; (iii) a DSL workbench as a service, equipped with editors, wizards, consoles, and dashboards for deployment and monitoring of self-adaptive cloud applications.

2 Related work and background

Self-aware management is becoming commonplace to address the scale, growth, and reliability of cloud applications. The authors of [IZM+17] propose a conceptual framework for analyzing the state-of-the-art and comparing practical characteristics, benefits, and drawbacks of self-awareness approaches used for cloud applications in different domains. A big challenge of self-aware and adaptive distributed systems on cloud is achieving self-protection as well as guaranteeing self-configuration, self-healing, and self-optimization. Aceto et al. [ABdDP13] examine current platforms and services for cloud monitoring pointing out their issues and challenges whereas the authors of [KA12] investigate testing models, recent research works, and commercial tools for cloud testing. However, the main

open issues emerging from the analysis of the literature related to the management of elasticity, dependability, and security of cloud applications are mainly about the limitations of flexibility and portability in a multi-cloud environment.

The OASIS TOSCA [OAS16] standard covers the cloud-portable automation of installation, configuration, and startup of conventional cloud applications. TOSCA is a declarative language that let model the distributed application topology independently from the particular target cloud infrastructure. It uses the following concepts: `Nodes` - nodes represent components of an application or service and their properties; example nodes are computer, network, storage (i.e. infrastructure-oriented), OS, VM, DB, Web Server (i.e. platform-oriented), functional libraries, or modules (i.e. applicative); (ii) `Relationships` - they represent the logical relations between nodes (e.g. hosted on, connects to), and describe the valid source and target nodes they link together; (iii) `Artifacts` - they describe installable and executable objects required to instantiate and manage a service; (iv) `Service Templates` - they group the nodes and relationships that make up a service's topology. In summary, TOSCA DSL allows describing a distributed cloud application at the infrastructure level in a portable way. Current TOSCA implementations target the main cloud provider infrastructures.

3 Outline of the solution

A DSL for self-adaptive cloud applications. We plan to overcome the limitations sketched in the section above by extending the TOSCA standard and implementations. The planned extensions of the TOSCA standard are about: (i) language traits for the installation, configuration, and setup of security and data protection provisions; (ii) language traits for structural and behavioral modeling of distributed applications, beyond the infrastructure level, at the application/service level; (iii) a policy description language that enables the definition of self-configuration, self-healing, and self-protection policies to be fulfilled at runtime. Security and data protection provisions to be automatically installed, configured and setup with the DSL deal with standard authentication, confidentiality, and integrity. Besides the DSL language traits to efficiently and conveniently express these security requirements, the implementation shall take into account security matters that are particular to cloud deployment. These concerns are about multi-sites communication, different hypervisors (code which manages the virtual machines), different hypervisor provided services that could have various security issues or expose security loopholes, different CPU/memory/storage. The TOSCA standard already provides a rich and flexible language for structural modeling (topology) at the infrastructure level. We plan to enrich the language with traits for structural modeling at the applicative level, by describing a distributed application as a graph of logical components connected by service dependency wires. At deployment time, this structure shall be installed, configured and setup at the applicative level too. This structural model is referenced by the self-configuration, self-healing and self-protection policies. Behavioral modeling shall leverage an existing standardized notation, such as the State Chart

XML (SCXML) [W3C14]. SCXML provides a powerful, general-purpose and declarative modeling language to describe the behavior of timed, event-driven, state-based systems. Therefore, in our context, state machines shall describe the external interactions between service components, explicitly showing the states of the conversation of each component with its wired interlocutors. Behavioral modeling will enable runtime checks and adaptation thanks to monitoring. The runtime policy language extension shall enable: (i) non-intrusive logging of events at the infrastructure and the applicative levels; (ii) non-intrusive monitoring (analysis of the logging stream) with slightly delayed evaluation of infrastructure events and distributed application behavior (asynchronous passive testing); (iii) non-intrusive active testing of the deployed application in a concurrent staging environment; (iv) intrusive active testing in the production environment to check application robustness and fault tolerance; (v) runtime installation, configuration and setup of components without service interruption, including new version deployment and version backtracking; (vi) automatic elasticity (scalability up and down); (vii) server failover (self-recovery); (viii) masking of transient network failures; (ix) circuit breaking at the application level; (x) generalized timeout management. An autonomic ops application manager shall implement and enforce these policies at run time.

The autonomic ops application manager. The classical autonomic architecture combines the managed application and an autonomic manager that oversees it. The abstract (platform independent) architecture of our ops manager includes (i) a supervisor, and (ii) a collection of concurrent feedback loops. Each feedback loop monitors and analyses a particular aspect of the managed application and its cloud environment and, if needed, plans and executes an adaptation process, driven by DSL policies, which brings the managed system to a new state. The abstract (platform independent) model of the feedback loop is the original MAPE-K generic architecture [IW15]. The supervisor controls the concurrent feedback loops and handles the interaction with the user, through the monitoring facility of the DSL workbench. There are no interferences between the managed application and the Ops manager, except for those performed by the adaptation processes. Significant concerns for the supervisor are the stability, accuracy, short settling-time, robustness, termination (no deadlock), consistency, scalability, and security of the adaptation processes. Important research questions are (i) the coordination of autonomous ops manager within cross-organizational distributed applications, and (ii) the DSL policy change at run time.

The DSL workbench. The DSL workbench (WB) is a Platform as a Service deployed on the cloud. It shall be accessible by the user via i) the Web-based Graphical User Interface, and ii) the WB API, a REST interface. The workbench is composed of three main layers: GUI, Processing, and Storage. The GUI Edit wizard allows: i) easy graphical drafting of the DSL artifacts (models, policies), ii) reverse engineering of existing legacy artifacts into the DSL ones, and iii) straightforward building of the DSL archives (sets of models and policies for an application to be deployed) and self-adaptive application releases (AR). The GUI Deploy wizard lets initiate and supervise the deployment of a AR on the target

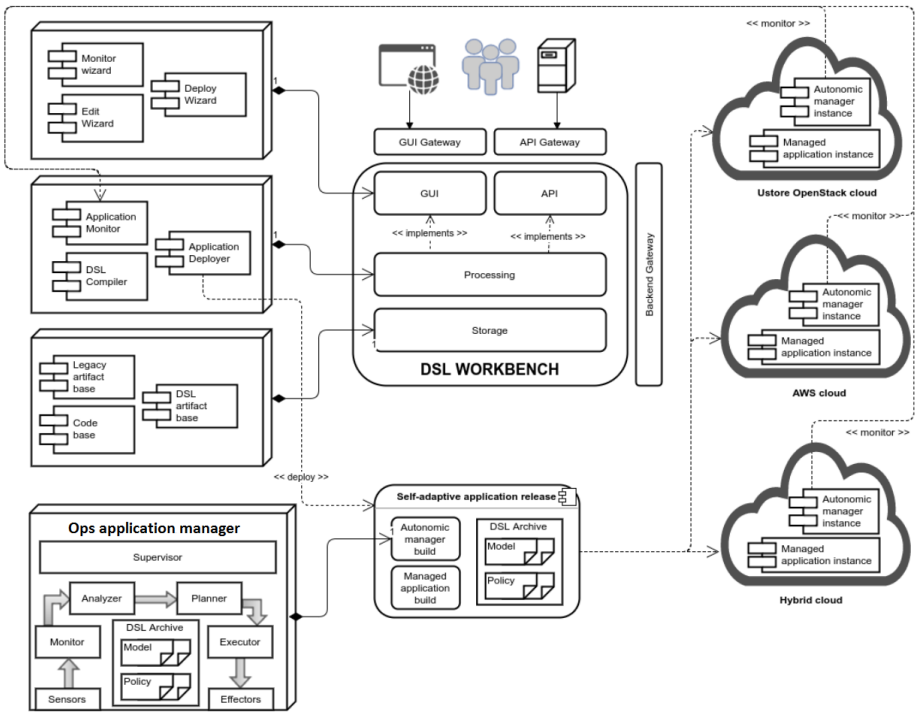

Fig. 1. General Architecture

cloud infrastructure. When deployed, the application ops manager interacts with the user through the GUI Monitoring wizard. The Processing layer is composed of three main components: the DSL Compiler, the Application Deployer, and the Application Monitor. The compiler builds the AR. The deployer installs, configures, and startups the application with the AR on the target (multi-)cloud infrastructure. The monitor implements the interaction with the deployed Ops Manager. The Storage layer contains: i) the Legacy Artifact Base, that stores the existing legacy artifacts such as Juju Charms, Kubernetes, Chef cookbooks; ii) the DSL Artifact Base that stores the DSL artifacts; iii) the Code Base that stores the codes of the Managed Application and of the Ops Manager. The general architecture is sketched in Figure 1.

Limitations of the approach. The main limitations of the proposed approach relate to: i) the fact that the proposed DSL is based on TOSCA and could collide or overlap with other emerging standards; ii) the inability or difficulty to integrate the proposed solution within all the available cloud provider infrastructures (i.e. Microsoft Azure, Google Cloud, etc.); iii) the lack of optimization methods for self-adaptive applications on cloud, except for the automatic scal-

ing up and down - the current project focuses on security, dependability, and fault-tolerance, not on self-optimization.

4 Real-world trials

We plan to try our solution with real-world applications and systems, in particular in the logistics and high-tech industries. In the logistics domain, decision making is distributed. All stakeholders make decisions locally and autonomously, so the most important challenge is to achieve collaborative decision-making in practice. The proposed solution can be adopted in the logistic domain for building a cloud-based logistics information platform, as a general exchange platform, where cloud services are composed to collect, classify, store, analyze, evaluate, publish (release), manage and control relevant information on inter-organizational logistics operations, processes, and management.

The proposed solution shall be applied to simplyTestify [sim], which is a geo-distributed, multi-instance and multi-tenant PaaS offering self-provisioning and pay-as-you-go test automation services. Even if simplyTestify core modules have been designed and implemented for cloud portability, the implementation of strong elasticity, dependability, security, and performance requirements is IaaS-dependent (the current version of simplyTestify runs on the Amazon Web Services public cloud). The DSL, the autonomic ops application manager, and the workbench shall allow the automatic installation, configuration, and startup of the PaaS and the policy-driven implementation of the mentioned requirements in a hybrid cloud including private and other public clouds, such as Microsoft Azure, Google Cloud, IBM Cloud, etc.

References

[ABdDP13] Giuseppe Aceto, Alessio Botta, Walter de Donato, and Antonio Pescap. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093 – 2115, 2013.

[IW15] Didac Gil De La Iglesia and Danny Weyns. MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems. *ACM Trans. Auton. Adapt. Syst.*, 10(3):15:1–15:31, 2015.

[IZM+17] Alex Iosup, Xiaoyun Zhu, Arif Merchant, Eva Kalyvianaki, Martina Maggio, Simon Spinner, Tarek Abdelzaher, Ole Mengshoel, and Sara Bouchenak. *Self-awareness of Cloud Applications*, pages 575–610. 2017.

[KA12] A Vanitha Katherine and K Alagarsamy. Software testing in cloud platform: A survey. *Int. J. of Computer Applications*, 46(6):21–25, 2012.

[OAS16] OASIS. TOSCA Simple Profile in YAML Version 1.0. OASIS Committee Specification 01, June 2016.

[sim] simplyTestify. http://simplytestify.com/pages/simplyTestify.

[W3C14] W3C. *State Chart XML (SCXML): State Machine Notation for Control Abstraction.* http://www.w3.org/TR/scxml/, May 2014.