The Journal of Systems and Software 000 (2016) 1–14

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Towards uniform management of multi-layered cloud services by applying model-driven development

Toni Mastelić[a,*], Andrés García García[b], Ivona Brandić[a]

[a] Institute of Software Technology and Interactive Systems, Vienna University of Technology, Favoritenstrasse 9-11/188, Vienna A-1040, Austria
[b] IBM Haifa Research Laboratory, Haifa, Israel

ARTICLE INFO

Article history:
Received 20 December 2014
Revised 30 September 2015
Accepted 2 January 2016
Available online xxx

Keywords:
Cloud computing
Cloud service model
Cloud management system

ABSTRACT

Cloud Computing started by renting computing infrastructures in form of virtual machines, which include hardware resources such as memory and processors. However, due to its popularity it gave birth to Everything-as-a-Service concept, where each service can comprise large variety of software/hardware elements. Although having the same concept, services represent complex environments that have to be deployed and managed by a provider using individual tools. The tools are usually used manually or specifically integrated for a single service. This requires changing an entire deployment procedure in case the service gets modified, while additionally limiting consolidation capabilities due to tight service integration.

In this paper, we utilize Model-Driven Development approach for managing arbitrary Cloud services. We define a metamodel of a Cloud service called CoPS, which describes a service as a composition of software/hardware elements by using three sequential models, namely Component, Product and Service. We also present an architecture of a Cloud Management System (CMS) used for automatic service management, which transforms the models from an abstract representation to an actual deployment. The approach is validated by realizing four real-world use cases using a prototype implementation. Finally, we evaluate its consolidation capabilities by simulating resource consumption and deployment time.

1. Introduction

Cloud Computing represents a new paradigm where arbitrary IT products, such as software applications, development environments and processors are integrated and offered as part of on-demand online services. According to National Institute of Standards and Technology (Mell and Grance, 2009), Cloud Computing defines three service layers, including Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS), also referred to as SPI service models. Unlike IaaS that usually offers a single distinct product, i.e., a virtual machine (VM), upper layers such as PaaS and SaaS provide services that are composed out of arbitrary software products such as database applications, web servers and storage platforms. This leads to an explosion of services such as Storage-aaS, Database-aaS, Identity-aaS and Computing-aaS, thus giving a birth to Everything-as-a-Service (XaaS) concept (Banerjee et al., 2011).

Building and managing such diverse services requires separate deployment tools (Forell et al., 2011) for each element that is part of the service, usually being distinguished by its deployment layer, e.g., VMs are deployed on an infrastructure layer using OpenNebula (Distributed Sys. Arch. Research Group, 2009), while a database is deployed using Docker (Docker, Inc., 2014). These tools are mostly used manually and separately for each layer, or in a best case scenario they are specifically integrated and customized for a single Cloud service. However, recent trends among Cloud providers show tendency towards multi-layered services, where each provider offers services spreading across several layers. One example is Microsoft, which offered a standard PaaS with their Azure (Microsoft Corporation, 2013), until they granted a VM access to their customers by introducing VM roles, hence reaching towards elements on an infrastructure layer. Another example is Amazon (Amazon Web Services, Inc., 2013a), which now covers all three layers with their broad assembly of services. Problem with such an approach is that a deployment procedure for each service is being built from the ground up, without reusing procedures from services that use the same elements. Moreover, a tight integration of services limits their consolidation capabilities as the service elements cannot be reused.

* Corresponding author. Tel.: +43158801188761.
E-mail addresses: toni@ec.tuwien.ac.at (T. Mastelić), angarg12@upv.es (A. García García), ivona@ec.tuwien.ac.at (I. Brandić).

http://dx.doi.org/10.1016/j.jss.2016.01.001
0164-1212/

In order to reuse existing deployment procedures and elements, Cloud service elements should be abstracted with a higher-level model (Ferrer et al., 2012) capable of describing any service with regards to its composition (Rodero-Merino et al., 2010). Furthermore, a Cloud Management System (CMS) must implement a uniform deployment procedure that utilizes this abstraction, while actual deployment is executed by lower-level tools specifically designed for a targeted piece of software. For example, a database application can be part of SaaS such as Facebook (Facebook, Inc., 2013), where a customer has no access to it. It can be part of PaaS such as Amazon RDS (Amazon Web Services, Inc., 2013a), where it is accessed by a customer, but managed by a provider. Finally, a database can be part of Amazon EC2 (Amazon Web Services, Inc., 2013a) as an example of IaaS, where it is both accessed and managed by a customer. In all three cases, a database application could be deployed with the same procedure using a different configuration.

In this paper, we introduce a uniform approach for deploying and monitoring arbitrary Cloud services. We utilize Model-Driven Development (MDD) (Mellor et al., 2003) for defining a Cloud service metamodel called CoPS, in order to get a uniform representation of a Cloud service. CoPS follows a Model-Driven Architecture (MDA) scheme proposed by OMG (Miller and Mukerji, 2003), which defines three levels of models that describe an environment on an abstract level, on a structural level and finally on an implementation level. We refer to these models as **S**ervice, **P**roduct and **Co**mponent, which form the abbreviation CoPS in a reverse order. The models can be sequentially transformed between each other going from an abstract representation to actual deployment. Additionally, CoPS allows model partitioning so models for individual service elements can be reused in other services.

We present a CMS architecture capable of managing Cloud services described with CoPS. Management of a service and its elements is performed on a structural level, since service components are represented through templates as black boxes, while transformation to an implementation model and final deployment is performed via plugins. Additionally, modularity of CoPS models allows the CMS to reuse templates and plugins for multiple Cloud services. Modules and interfaces of the architecture are described using Unified Modeling Language (UML) (Rumbaugh et al., 1999).

A prototype of the architecture is implemented using Cloudcompaas (García et al., 2010), a Cloud manager framework for dynamic management of Cloud resources, and M4Cloud (Mastelic et al., 2012), a plugin-based monitoring tool capable of monitoring arbitrary metrics. The prototype is used for validating both CoPS metamodel and the CMS architecture by realizing four use cases based on real world scenarios. Additionally, we simulate a resource utilization and deployment time in order to show benefits of a modular service composition provided by our approach. Results show that our approach provides better consolidation capabilities by reducing resource consumption over 10%, as well as speeding up a deployment time by 4x.

The rest of the paper is organized as follows. Four use cases used throughout the paper are described in Section 2. Section 3 gives a motivation for the approach taken in this paper. Section 4 introduces the CoPS model. Section 5 provides a detailed description of the architecture, while Section 6 describes the implementation of its prototype. Section 7 gives validation of the CoPS metamodel and the architecture by realizing the three use cases. Section 8 presents simulation results showing resource utilization and deployment time. Section 9 describes relevant related works. Finally, Section 10 concludes the paper and proposes the future work.

2. Use case scenarios

We depict four use cases based on real world scenarios to emphasize issues related to management of Cloud services.

- **Online Course**: Students of an online course require a specific software stack to complete exercises of the course, similar to Strawberry Canyon LLC (2013). The software stack includes a large number of tightly coupled Python libraries, some of them specific to the course topic and not widely available. In order to avoid a cumbersome operation of manually installing the libraries, the course offers an online service for instantiating a VM with a customizable software stack.
  This is a basic IaaS scenario where a provider such as Amazon (Amazon Web Services, Inc., 2013a) offers VMs. Additionally, the provider also offers ready to use VM images with preinstalled software (Canonical Ltd., 2013). Another examples of such services are AWS Marketplace (Amazon Web Services, Inc., 2013b) and Vagrantbox (Rushgrove, 2013), which offer preconfigured VM images.
- **Genomics**: A group of scientists want to run their scientific applications in a Cloud. They utilize distributed genomic applications that work on large datasets similar to CloudBlast application (Matsunaga et al., 2008). However, since not every instance works over a complete dataset, but only a small subset, transferring data to each machine incurs a large overhead in terms of time and cost. Therefore, they want to utilize an arbitrary number of VMs with access to a third party online BioDatabase such as GenBank (Benson et al., 2009) or EMBL Nucleotide Sequence Database (Kanz et al., 2005) allowing customers to retrieve only a subset of genomic data they are interested in.
  This use case represents a collocation of VMs, which are on an infrastructure layer, with a database offered on a software layer, thus providing an unique platform for running genomic applications. A similar scenario can be found in Amazon Web Services, Inc. (2013a) where a customer can utilize additional hardware and software products along with deployed VMs. Heroku also provides a large set of add-ons (Heroku Inc., 2013), i.e., software products that can be referenced within customer applications.
- **Web hosting**: A web developer wants to migrate a web application to a Cloud in order to benefit from a highly available and scalable environment. However, the customer does not want to manually configure and manage the environment, but rather have it all done automatically.
  This use case represents a traditional PaaS where a customer deploys an application on a targeted platform. Google App Engine (Google Inc., 2013) enables deploying Java and Python web applications on a managed Cloud platform that is completely transparent to a customer. Heroku Inc. (2013) offers a Ruby on Rails environment deployed on top of Amazon EC2. The platform is completely managed by the provider, while customers only interface with provided runtimes, databases and add-ons. Windows Azure (Microsoft Corporation, 2013) supports deployment of web and non-web applications in a Windows runtime for a variety of programming languages.
- **Email**: Private and business customers require online access to an email service. The customers pay the service per registered user and the amount of used storage, while not considering the implementation details of the service.
  This is a standard SaaS example where a provider is responsible for a service infrastructure, including entire hardware and software stack. Customers access the service through a web client and interact with the software over its web interface. Entire management of the service is done by a provider, including software updates, environment scaling and security. Examples

Please cite this article as: T. Mastelić et al., Towards uniform management of multi-layered cloud services by applying model-driven development, The Journal of Systems and Software (2016), http://dx.doi.org/10.1016/j.jss.2016.01.001