Institute of Architecture of Application Systems

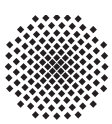
Improving Portability of Cloud Service Topology Models Relying on Script-Based Deployment

Johannes Wettinger, Oliver Kopp, Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{wettinger, kopp, leymann}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Wettinger2013,
  author    = {Johannes Wettinger and Oliver Kopp and Frank Leymann},
  title     = {Improving Portability of Cloud Service Topology Models
               Relying on Script-Based Deployment},
  booktitle = {Proceedings of the 5th Central European Workshop on
               Services and their Composition (ZEUS 2013)},
  year      = {2013},
  publisher = {CEUR-WS.org}
}
```

Improving Portability of Cloud Service Topology Models Relying on Script-Based Deployment

Johannes Wettinger, Oliver Kopp, and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{wettinger, kopp, leymann}@iaas.uni-stuttgart.de

Abstract Portability is key for services running in the Cloud to prevent vendor lock-in. Today, many Cloud services are portable and can thus be moved from one Cloud provider to another. However, the management of these services is often bound to provider-specific management tooling. Thus, the way of management of a particular Cloud service may completely change when moving it to another Cloud provider. This paper presents concepts to improve the portability of Cloud service topology models that are deployed and managed using scripts. We highlight the challenges of a semi-automatic procedure to generate portable TOSCA-compliant topology model components based on Juju topology model components.

Keywords: portability, service topology, topology model, script-based deployment, Cloud computing

1 Introduction

Reducing the costs of infrastructure and service management is one of the most important aspects of Cloud computing because traditional IT service management is costly. This goal is achieved by automating the whole management of services running in the Cloud. Management of Cloud services is not limited to deploying and decommissioning service instances; it includes several management tasks that need to be performed once a particular service instance has been deployed. As an example, the service instance has to scale up and down depending on the current workload. Today, Cloud providers offer proprietary tooling to automate Cloud service management such as "CloudFormation" and "Auto Scaling" provided by Amazon Web Services[1]. The learning curve is flat because these tools are easy to use. However, when the service is moved to another Cloud provider the management tooling is different. Thus, the service may be managed in a completely different manner. The service itself may be perfectly portable, so it can be moved from one Cloud provider to another. However, this may not be true for the service management. This is why portability is essential for services running in the Cloud, especially when it comes to service management.

[1] Amazon Web Services: http://aws.amazon.com

To achieve management portability, this paper provides two key contributions: (1) an approach to generate standard-compliant topology model components and (2) concepts to improve portability of these generated components.

2 Background

We assume that the structure and management behavior of a Cloud service is specified using a service topology model consisting of several topology model components. As an example, two topology model components may be part of a topology model for deploying and managing a Web application: an "Apache Web server" and a "MySQL database server." A topology model component contains scripts that are typically implemented using a scripting language such as Python or Perl. These scripts realize the management actions that can be performed regarding a service instance of the particular topology model such as deploying and updating its components. We focus on *service deployment* as one of the most important management tasks, based on topology models. Today, there are existing topology model components publicly available that can be used to deploy and manage services in the Cloud. A prominent example is Juju[2]. The community shares more than one hundred topology model components as open source software. Such a component is called a "charm" and can be combined with other "charms" to create a service topology model that can be instantiated and managed in the Cloud. The core of a charm is a set of scripts to enable automated management of a particular service instance. However, these scripts are bound to Ubuntu Linux and thus are not portable. There are standardization efforts going on in the field of model-driven Cloud management that are focusing on management portability: the Topology and Orchestration Specification for Cloud Applications (TOSCA)[3] is an emerging standard supported by a number of prominent companies in the industry such as IBM, SAP, and Hewlett-Packard. TOSCA enables the specification of portable topology models and portable topology model components. However, an ecosystem including an active community sharing topology models and topology model components based on TOSCA is still missing.

3 Generating Standard-Compliant Topology Model Components

One goal of our work is to bring together the standardization efforts of TOSCA enabling management portability with Juju's growing ecosystem and active community. The first step to achieve this goal is outlined in this section: transforming topology model components published by the Juju community to TOSCA-compliant topology model components. Both TOSCA's and Juju's topology models basically specify graphs consisting of nodes and relations between nodes

[2] Juju: `http://juju.ubuntu.com`
[3] TOSCA: `http://www.tosca-open.org`

to define the structure of a Cloud service. In TOSCA, both relations and nodes are explicitly modeled as separate topology model components, whereas Juju specifies nodes as topology model components only. Consequently, two major steps have to be performed: (1) a TOSCA-compliant topology model component has to be generated for each Juju charm; as a result, each node that can be modeled using Juju, can be modeled using TOSCA, too. However, the relations between these nodes cannot be modeled in TOSCA because the corresponding topology model components are missing. (2) Thus, additional TOSCA-compliant topology model components have to be generated for each relation that can be implicitly modeled using Juju.

As an example, for the Juju charms "WordPress application" and "MySQL database server," two corresponding topology model components are generated that can represent nodes in a TOSCA topology model (service topology). For the relation "WordPress application connects to MySQL database server," which can be implicitly modeled in Juju, a separate topology model component is generated that can represent the corresponding relation in a TOSCA topology model.

4 Improving Portability of Generated Topology Model Components

The generated topology model components as described in Section 3 are TOSCA-compliant and thus follow an emerging standard. Service topology models using these components can be deployed and managed using an arbitrary TOSCA engine such as OpenTOSCA[4] or IBM SmartCloud Orchestrator[5]. This is already an improvement of portability because the original topology model components shared by the Juju community can be processed by the Juju engine only. However, the scripts inside the topology model components are still restricting the portability in two ways: (1) the scripts use a set of commands and environment variables that are available on each virtual machine managed by Juju. (2) The scripts are designed to be executed on Ubuntu Linux; as a result, their execution fails on other Linux variants and other platforms.

The first restriction can be compensated by generating wrapper scripts that prepare the execution environment and then call the actual scripts originating in Juju charms. These wrapper scripts receive their input from the TOSCA engine and expose commands and environment variables that are used by the actual scripts. The second restriction is a greater challenge: the generated topology model components have to be refined to further enhance their portability either by improving the existing scripts so they also run on other platforms or by creating and attaching additional scripts to support other platforms. These additional scripts can be created by copying an existing script and semi-automatically adapting it to be executable on another platform. This alternative realizes separation of concerns and thus is the preferred one in contrast to directly modifying an existing script.

[4] OpenTOSCA: `http://www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php`
[5] IBM SmartCloud Orchestrator: `http://ibm.co/CPandO`

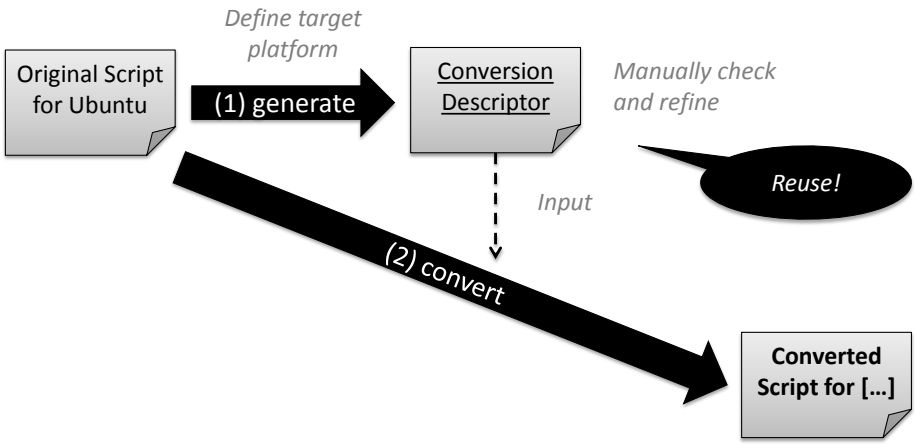


Figure 1. Conversion descriptors as a means of improving the portability of scripts

We are currently designing a semi-automatic, modular, and extensible procedure to convert a script that is implemented for a specific platform to be executed on another one. Figure 1 presents an example to show the basic concept of the procedure: based on the original script that is bound to a specific platform such as Ubuntu Linux and the definition of a particular target platform a conversion descriptor gets generated. It specifies the changes that are required to make the original script executable on a particular target platform such as Red Hat Linux. Then, the generated conversion descriptor gets checked and refined manually because some of the specified changes may be incorrect. Additionally, changes that are actually required may not represented at all. Based on the refined conversion descriptor the actual conversion procedure gets performed to create a new version of the script that is executable on the target platform. The purpose of generating a conversion descriptor is twofold: first, it enables a manual review regarding correctness and completeness. Second, the conversion descriptor can be reused and further refined for future versions of the original script.

The concepts described in Section 3 and Section 4 enable the creation of portable topology models based on TOSCA using the generated and refined topology model components owning a high degree of portability. In future, we focus on reusing these concepts to generate additional topology model components originating in communities of configuration management tools such as Chef[6] or Puppet[7].

Acknowledgments The research leading to these results has partially received funding from the 4CaaSt project part of the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258862. Further, this work was partially funded by the BMWi project CloudCycle (01MD11023).

[6] Chef: `http://www.opscode.com/chef`
[7] Puppet: `http://www.puppetlabs.com`