2014 IEEE International Conference on Cloud Engineering

Cloud application portability with TOSCA, Chef and Openstack

Experiences from a proof-of-concept implementation.

Gregory Katsaros
FZI Forschungszentrum Informatik
Berlin, Germany
Email: katsaros@fzi.de

Michael Menzel
FZI Forschungszentrum Informatik
Karlsruhe, Germany
Email: menzel@fzi.de

Alexander Lenk
FZI Forschungszentrum Informatik
Berlin, Germany
Email: lenk@fzi.de

Jannis Rake-Revelant
Telekom Innovation Laboratories
Berlin, Germany
Email: jannis.rake-revelant@telekom.de

Ryan Skipp
T-Systems International GmbH
Bellville, South Africa
Email: ryan.skipp@t-systems.com

Jacob Eberhardt
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: jacob.eberhardt@student.kit.edu

***Abstract*—As the sector of SaaS is being evolved, the need of portability and effective orchestration of Cloud-enabled applications over virtualised infrastructures is more apparent. In this paper we present a proof of concept (PoC) project together with industry partners which deals with the management of portable Cloud applications. Experimentation with a multi-tier Cloud application use case gives insights into the capabilities of the TOSCA specification (version 1). To realize support for a Cloud environment based on OpenStack and Opscode Chef, an environment that allows the execution and management of TOSCA service topologies has been developed. For the evaluation of the implemented system, we orchestrate the un-/deployment of the multi-tier Cloud application use case and in the end we discuss the results and experiences of this exercise.**

***Keywords*—TOSCA, service orchestration, Cloud applications, Chef, Openstack**

I. Introduction

Enterprise IT operators derive three main benefits from the adoption of Cloud infrastructure technology: (a) instant availability of Cloud services, (b) Cloud services can be facilitated to gain elastic software behavior and billed per use (metered resource usage), and (c) provision existing service offerings from a market with multiple competing providers.

Cloud infrastructure services allow the operation of software applications on virtual machines managed in public or private Cloud environments. Virtualization technology is the enabler of such functionality which facilitates the automated creation of virtual machines and discloses remote interfaces for virtual machine management. However, there are necessary prerequisites for realizing the expected Cloud benefits. Software applications often consist of multiple components, thus, a Cloud service must allow deployments of a software application over multiple virtual machines. Additionally, software applications might be operated on virtual machines of various Cloud providers. Virtual machines occupied in such a setup can be part of a private Cloud or a public Cloud. Unlike public Clouds, private Clouds are operated and maintained in a companys own data centers. Setups with a mix of public and private Cloud infrastructure are referred to as hybrid Clouds.

A relocation of software applications between substitutable services of competing Cloud providers is not unusual.[1] Similarly, distributing components of a software application across multiple providers or a hybrid setup is not uncommon. Whenever an application or parts of it need to be deployed over multiple services or in multiple copies, a repetitive deployment task occurs. Obviously, there is a need to be able to package and transport applications and related parameters within or across different Cloud environments. This portability characteristic is an essential operational feature for contemporary Cloud applications. While migrations to Cloud services have already been investigated for the application and data layer [2][3][4], Cloud application portability has been only recently explored [5]. One activity that pursues creating a standard for portability in the context of Cloud computing, which brought together various important industrial partners as well as people from academia, is the definition of the Topology and Orchestration Specification for Cloud Applications (TOSCA) [6]. This effort is supported by OASIS (Organization for the Advancement of Structured Information Standards) and is sponsored by important companies of the ICT sector such as IBM, CA Technologies, Hewlett-Packard, Red Hat, SAP and others.

A team led by T-Systems, Telekom Innovation Laboratories, and the FZI Research Center for Information Technology (FZI[1]) carried out a proof of concept (PoC) project to investigate the current state of the art in portability of Cloud applications. A Cloud-enabled Web application is defined as a use case and is the subject of a portability project. The TOSCA specification is the basis and the technological driver for the implementations in this PoC project.

The PoC project leverages the use case Web application to investigate how a Cloud application can be ported to a certain Cloud infrastructure. The activities of the PoC

[1] http://www.fzi.de

978-1-4799-3766-0/14 $31.00 © 2014 IEEE
DOI 10.1109/IC2E.2014.27

project included a state-of-the-art analysis and a selection of the appropriate technologies, i.e., the TOSCA specification. Furthermore, insights regarding capabilities of the TOSCA specification are generated. A modeling of the use case application in compliance with the specification shows where limitations or openness are hindering. Moreover, an execution environment named TOSCA2Chef is developed that allows the automatic deployment and configuration of software components in a Cloud application. TOSCA2Chef is able to deploy Cloud application topologies defined in TOSCA to OpenStack Clouds by employing the Opscode Chef configuration management software and BPEL processes. Within our investigation, TOSCA2Chef serves as the basis to evaluate the current state-of-the-art in regards to portable Cloud applications.

In the following section, we first introduce the state-of-the-art in the field of Cloud application portability. Section III describes the testbed environment and technologies fundamental to the presented activity. In section IV, a use case web application is motivated and defined. Furthermore, section V elaborates on the topology of the use case application modeled in compliance with the TOSCA v1 specification. An implementation of an execution environment that allows automated deployments of the use case application is explained in Section VI. Section VII presents the findings of experiments to capture current state of the art of portability for the use case. Finally, section VIII concludes and discusses the contributions and findings.

II. State-of-the-art

Cloud portability is the ability to move applications and its associated data between Cloud environments. The goal of Cloud portability is to automate the configuration, coordination and management of software and software interactions in Cloud infrastructures. This task is sometimes complicated as it involves interconnecting processes running across heterogeneous systems in multiple locations, usually with proprietary interfaces.

To effectively succeed with Cloud portability, the development of tools, frameworks and specifications that facilitate the Cloud orchestration and allow migration of applications, is necessary. Lately there have been several initiatives and efforts so much in research [7] [8] [9] [5] but also in industry [10][11] dealing with Cloud application orchestration and portability or tools and specifications to describe service topologies. Most of them are independent projects supported by only a part of industry or academic organizations and they do not reach a high level of adoption. There are also others that receive the support from bigger part of industry and manage to become official standards or be adopted by important companies of the field. In the following paragraphs, we will elaborate on some selected technologies and solutions, namely CAMP, TOSCA, CloudFormation and Heat.

A. CAMP

A first approach to enhance interoperability is to unify the management interfaces across multiple Cloud infrastructures, which is what the standard introduced here aims at: Cloud Application Management for Platforms (CAMP)[12] is a specification designed to ease management of applications across platforms offered as a service (PaaS). This OASIS proposed standard specifies a RESTful generic self-service application and platform management API, which is language, framework, and platform neutral. It is limited to PaaS offerings and aims to enhance interoperability of the interfaces provided by such platforms and with that PaaS adoption in general by reducing vendor-lock in. Furthermore the specification facilitates the creation of services interacting with complying platforms.

The specified CAMP API consists of a resource model and a protocol to remotely manipulate these resources. The resources contained in the model are divided into a Platform, Assemblies, Platform Components and Application Components. A Platform resource describes a PaaS offering as a whole. It references all applications on the platform and allows discovery of Platform Components. Platform Components (e.g. database service, web server) are PaaS vendor provided and can be used by invoking applications. An Application Component is an artifact (e.g. source code, resources, metadata), which can make use of other Application Components and depends on Platform Components. As depicted in Figure 1, an Assembly is a resource representing an application by referencing the components it is composed of thereby allowing runtime management.

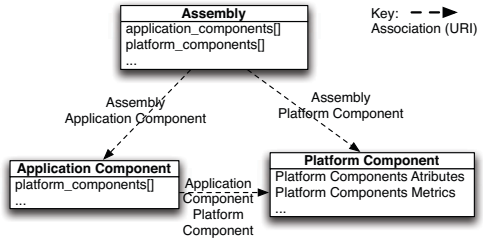

Fig. 1: CAMP Ressource Relationships - Assemblies and Components. Source: [12]

B. TOSCA

While CAMP offered only limited functionalities to deal with the aspects of portability and orchestration, there is one industrially-endorsed standardization effort in the area of application topology specification that proposes a more comprehensive approach. The Topology and Orchestration Specification for Cloud Applications (TOSCA) [6] aims to leverage portability of application layer services between various Cloud environments. Software components and their relationships (topology model), as well as management procedures to orchestrate operational behavior (e.g. deployment, patching, shutdown), can be described in an interoperable way using the specified XML-based language. As depicted in Figure 2, TOSCA introduces Service Templates encapsulating a Topology Template describing an applications topology model as well as plans defining manageability behavior using process models (e.g. BPEL, BPMN).

This abstract description can be mapped to concrete infrastructure by Cloud providers, which enhances usability and hides complexity from the user of the Cloud application.

Regarding available tools, only recently (end of September 2013) the first open source available software was published

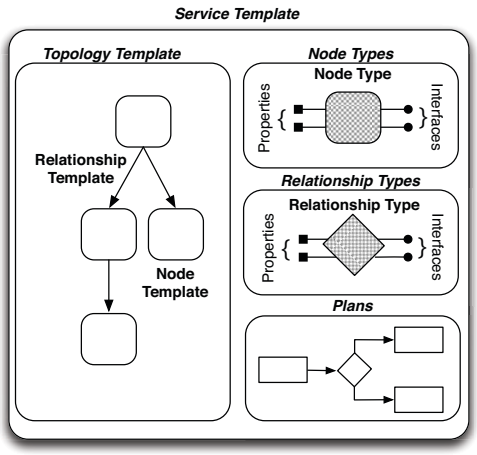

Fig. 2: Structural Elements of a Service Template and their Relationships. Source: [6]

through the OpenTOSCA initiative [13][14]. Specifically a modeling tool called Winery [15][16] was published, through which one can define a Cloud application topology using a Web GUI. A first version of the TOSCA container infrastructure was also released.

It has to be noted though that development of this PoC as well as the implementation of our complete software solution, have been performed before the release of the OpenTOSCA tool-set. Therefore, we had to design and implement our own environments and software components, at that point based on the first version of the TOSCA specification documentation.

C. AWS CloudFormation

In contrast to the previous general specifications, Amazon Webservices (AWS) CloudFormation [17] is an orchestration approach proprietary to the Amazon Infrastructure. A JSON-compliant template describes a set of related AWS resources (EC2 or Elastic Load Balancer instances, S3 buckets) called a stack. This stack can then be managed (created, updated and deleted) as an entity. AWS CloudFormation is declarative, which means the platform automatically resolves dependencies and orchestrates the creation of the AWS resources declared in the template and connects them afterwards. There is no need to specify management processes (e.g. deployment processes).

D. HEAT

OpenStack [18], one of the most popular Cloud middleware software stacks, introduces its own orchestration approach: Heat [19], is the main project of the OpenStack Orchestration program, enabling declarative infrastructure provisioning while being portable between OpenStack Clouds. It provides a cross-compatible AWS CloudFormation implementation for OpenStack and introduces an advanced template language based on YAML (strict superset of JSON).

III. Testbed and Fundamental Technologies

The end-to-end management of Cloud-enabled service topologies using TOSCA demands a framework of components that will allow the translation of a topology defined though the TOSCA specification to the commands that will actually realize the deployment of instances and software packages to the Cloud. The testbed we used in our work was hosted in the T-Labs OpenStack Testbed, in an isolated network partition. Existing OpenStack [18] based infrastructure was leveraged using the Opscode Chef [20] platform to perform, manage and automate virtual machine (VM) deployments and the software package installations. The management of both systems is achieved through the Knife client extended with an OpenStack plug-in. Therefore, our proposed solution combines the technologies of Chef, Openstack and BPEL in order to evaluate an end-to-end, realistic scenario. In the following paragraphs, we briefly introduce those technologies before proceeding to the presentation of our use case.

A. OpenStack

For the infrastructure provisioning on the testbed, we relied on OpenStack [18], Grizzly release. It is a free open source IaaS Cloud platform, which controls and manages compute, storage, and network resources aggregated from multiple physical compute-nodes. For flexible management, monitoring and on-demand provisioning of resources, a web interface and APIs are available. As described in [21], OpenStack consists of three main projects: (a) OpenStack Compute, a scalable compute provisioning engine, (b) OpenStack Object Storage, a fully distributed object store, (c) OpenStack Imaging Service, an image registry and delivery service, and (d) Quantum Network Service, a tenant-facing API for defining network connectivity an addressing in the Cloud.

B. Opscode Chef

Configuration management was handled using Opscode Chef [20]. It is an open source software, operating on a client-server model, enabling to describe and manage system configuration using a Ruby based domain-specific language (DSL). Chef manages so called nodes, which can be physical or virtual machines running a Chef client. This client performs the automation tasks the specific node requires. The nodes register at a server, which then provides recipes defining these automation tasks and assigns roles. Cookbooks are used to organize related recipes, which are basically Ruby scripts, and supporting resources (e.g. installation files). Roles contain lists of recipes, which are then executed by the Chef client upon retrieval from the server, leading to the desired configuration. Furthermore Chef provides a command line tool called Knife, allowing to interact with the Chef server (e.g. install Chef client, upload cookbooks). An extended Knife-Openstack client provides the means to instantiate and configure VMs with a single interface.

C. BPEL & ODE

For the definition of management plans contained in TOSCA service templates, we chose the Business Process Execution Language (BPEL). This XML-based language allows orchestration of web services by defining executable business processes as specified in the OASIS standard [22]. As a BPEL-engine, allowing us to execute these plans, the open source BPEL-engine Apache Orchestration Director Engine (ODE) was used [23].

IV. Use Case Definition

For the realization of this proof of concept project a specific multi-tier application service topology had to be designed as the use case through which we would evaluate the related technologies. As a requirement for an appropriate investigation an application not too simple (only one or two components) and not too complex had to be found. The number of components and inter-connections should stress the capabilities of existing portability technologies but not exaggerate the efforts of the PoC. Moreover, a commonly used application stack with well understood relationships and dependencies was sought. As a result of discussions with industry partners of the PoC a basic 3-tier web application has been settled for as the use case. In detail, the use case web application comprises a Load Balancer component implemented through the HAProxy open source software package [24] and two application servers implemented through Apache Tomcat [25], hosting a Demo Web Application for demonstrating the operation. On the data layer we introduced a database server through a MySQL implementation that keeps some dummy data for the demonstration. There are inter-connections between components of the web applications. The graphical representation of the topology is presented in Figure 3.

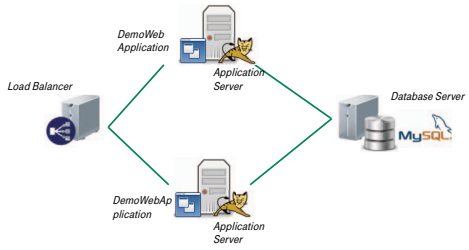

Fig. 3: The selected 3-tier web application use case.

Our objective is to successfully deploy and un-deploy all the software components of the application as well as the VM instances which will host those components. The un-/deployment process should be done in an automated manner based only on the topology description. Therefore, we used the TOSCA specification to define all the installation and configuration details for all components as well their inter-relationships.

V. Modeling the use case with TOSCA

Following the concepts of the TOSCA specification, a definitions part and a topology description part need to be modeled. For the definition of the resources available in the environment, a series of NodeTypes, RelationshipTypes and ArtifactTypes have been defined using the TOSCA specification. These definitions help to capture all entities of the use case application. Given the definitions, a topology description (TopologyTemplate) can define inter-relations between components referred to as nodes in TOSCA. Since all *Type definitions are actually reusable according to TOSCA, they could populate a pool of resources available to other application developers, too. The NodeType definitions of the use case application are presented in Figure 4.

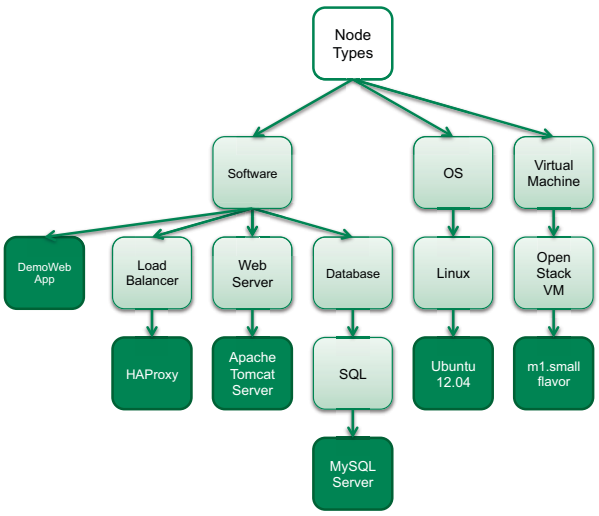

Fig. 4: Nodetype definitions.

The TOSCA specification supports an inheritance functionality (*DerivedFrom* tag), which allows us to design application components and define several layers and types. In that context, we distinguish the nodes into three basic categories: Software, Operating System (OS) and Virtual Machine. Furthermore, we define additional abstractions for each category such as Database, SQL or Linux etc., allowing the definition of generic interfaces and operations. The dark green NodeTypes (Figure 4) are the final types that are also implemented (through a NodeTypeImplementation element) and compose our use case application deployment. Every NodeTypeImplementation declares a DeploymentArtifact, which is the one that actually includes the scripts (in our case Chef Roles and Recipe names) in order for the node to be instantiated during the deployment. In Figure 5, we visually present the definitions of the NodeTypeImplementations and DeploymentArtifacts for our use case application.

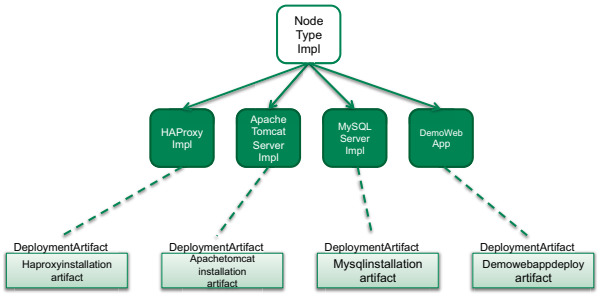

Fig. 5: NodeTypeImplementation diagram and DeploymentArtifact definitions.

The following part of example XML code describes the implementation for the NodeType HAProxy. The *RequiredContainerFeature* functionality is used in order to define the environment in which this implementation is valid - in our case the OpenStack and Chef testbed. In addition, we define a DeploymentArtifact with the reference *haproxyinstallationtem-*

plate which includes the implementation scripts or commands. As can been seen in the following lines of code, the artifact includes the declarations of the roles and/or recipes necessary for the Chef bootstrapping of that very node.

```
<NodeTypeImplementation name="HAProxyImpl"
nodeType="HAProxy">
      <RequiredContainerFeatures>
            <RequiredContainerFeature
            feature="http://telekom.de/openstackCloud/" />
            <RequiredContainerFeature
            feature="http://telekom.de/opscodechef/" />
      </RequiredContainerFeatures>
      <DeploymentArtifacts>
            <DeploymentArtifact name="haproxyinstallationartifact"
            artifactType="chefinstallation"
            artifactRef="haproxyinstallationtemplate" />
            <DeploymentArtifact name="haproxyconfigartifact"
            artifactRef="haproxyconfigtemplate"
            artifactType="chefconfig" />
      </DeploymentArtifacts>
</NodeTypeImplementation>
<ArtifactTemplate id="haproxyinstallationtemplate"
type="haproxyinstallation\">
      <Properties>
            <roles>
                  <chef:role name="haproxy" />
            </roles>
            <recipes>
                  <chef:recipe name="apt" />
            </recipes>
      </Properties>
</ArtifactTemplate>
```

In the use case application, the HAProxy component is balancing the load between the two DemoWebApp deployments, which in turn communicate with the MySQL server to retrieve the relevant data. Based on the general *HostedOn* and *Communication RelationshipTypes*, we have defined specific ones in order to capture the relation of each component with each other and define specific parameters for the realization of the topology. In Figure 6, we present all the RelationshipTypes and their hierarchy. In the same way that nodes are instantiated (through artifacts), the relationships can also be configured. In that context, ImplementationArtifacts can be defined to capture certain parameters of a relationship between two nodes.

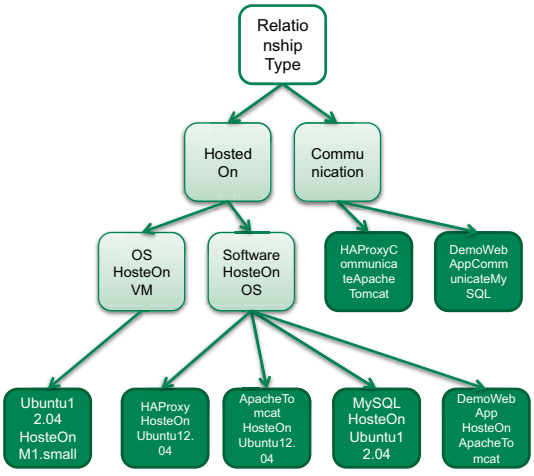

Fig. 6: Relationship types definitions.

A topology description (TopologyTemplate) was defined to capture the relationships between nodes with multiple instances for Apache Tomcat NodeTypes. In Figure 7, we can see the TopologyTemplate of the use case application containing the software components (HAProxy, Apache Tomcat and MySQL server) to be hosted on an Operating System that is hosted on an OpenStack Virtual Machine instance. The DemoWebApplication we used to demonstrate the operation is hosted on an Apache Tomcat application server.

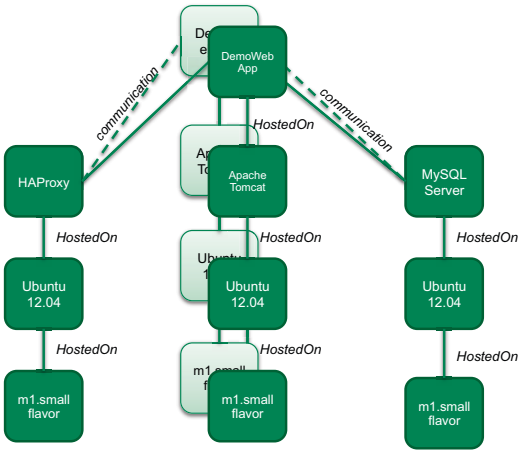

Fig. 7: Use case Service Topology design.

VI. TOSCA2Chef Execution Environment

The goal of the work is to test the actual portability features of TOSCA, from modeling phase of the use case to a web application running on at least one Cloud infrastructure. An execution environment that transforms a TOSCA model into a running application is not available[2]. Also, according to the TOSCA specification un-/deployment plans have to be defined that orchestrate the deployment of the use case topology. TOSCA plans are defined as process models, i.e., a workflow of one or more steps, in either BPMN or BPEL. Such plans have to be supported by an execution environment as well. The core entity of the proposed architecture is the TOSCA2Chef execution environment: a set of components and services that evaluate a TOSCA document, extract the information regarding the described application and trigger the necessary command executions towards the Chef Knife client. The Knife client interacts with interfaces of the Chef server and OpenStack compute Cloud to assign compute resources to the application and trigger deployment or configuration tasks.

The TOSCA2Chef execution environment incorporates two basic operations: (a) the parsing of the TOSCA document in order to perform the deployment of a given topology, and (b) the execution of the deployment (or un-deployment) TOSCA plans. Parsing of TOSCA documents and execution of the necessary commands towards the test bed is being managed through the TOSCA Container web service. The definition and execution of BPEL processes to orchestrate un-/deployment is realized using the Apache ODE BPEL engine [REF]. Figure 8 depicts the architecture of the TOSCA2Chef execution environment.

[2]At the time of the PoC implementation.

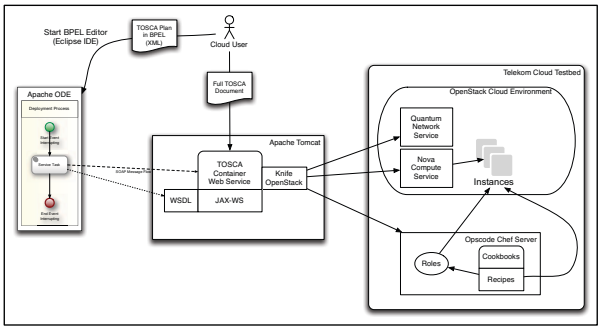

Fig. 8: High level architecture of the TOSCA2Chef execution environment.

For the effective management of the TOSCA topology descriptions, we introduce web service interfaces facilitating the file management and plan execution. The interfaces offered by the TOSCA Container web service are presented in Figure 9.

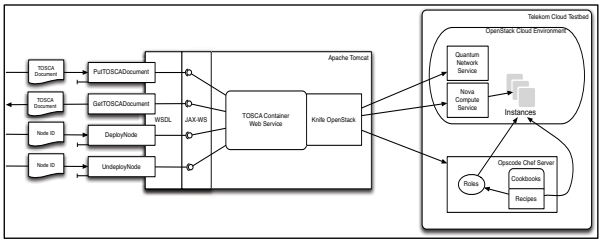

Fig. 9: TOSCA Container web service.

To transform a parsed TOSCA XML document into running VMs in the test-bed, an intermediary data model was required that reflects a TOSCA topology in terms of Chef and OpenStack concepts. The intermediary model builds the basis to deploy - or un-deploy - VMs via the Knife-Openstack client. Given that a TOSCA model derives from a TOSCA XML document, a transformation logic can derive an intermediary data model. Since no tools were available for parsing a TOSCA document[3], a library for the XPath XML query language was employed to replace a missing TOSCA parser or library. Information from a TOSCA XML file is extracted with a set of XPath queries and stored into the intermediary data model. An execution engine based on this model orchestrates the deployments.

In order to enable the deployment of the described application to a specific test-bed environment, an intermediary data model (Figure 9) must include some domain specific entities that are relevant with the technologies used in the Cloud test bed. To this end, our data model captures information regarding the Openstack VM images and flavors as well as the Chef recipes and roles. Figure 10 depicts the data model that was introduced. It captures domain specific infrastructure and topology data, consisting of a Node with a Flavor, Image, Attributes and a Runlist of Recipes and Roles. The extension of the developed system to support different IaaS providers (e.g. AWS) would require the introduction of a different data model that could capture the new, domain-specific attributes (e.g. AMIs). In addition, we would have to adapt the TOSCA parsing code in order to fill the new data model in its transformations. Furthermore, the plan execution operation would have to be adapted to be able to base its execution logic on the new model.

[3]at the moment of the PoC implementation

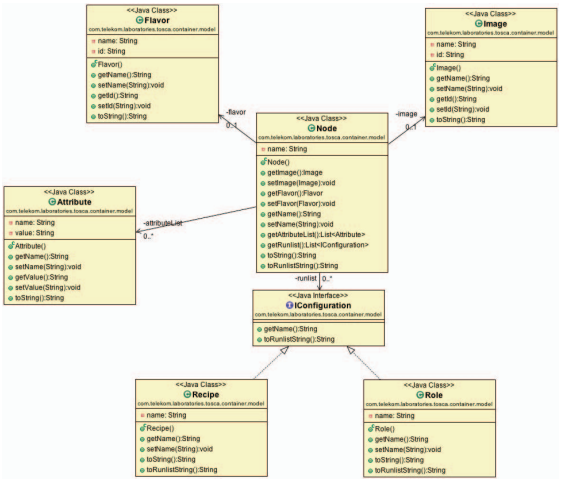

Fig. 10: Intermediary, domain specific data model.

Moreover, two business processes were defined within the TOSCA document, responsible for the deployment and un-deployment actions. The processes were defined in BPEL XML notation within the plans section of the TOSCA document. Both BPEL processes could only be modeled with the knowledge of the TOSCA2Chef web service interfaces. Either BPEL process interacts with the web service to deploy or un-deploy nodes from the use case application topology. In Figure 11 we present the graphical representation of the deployment process for our use case application. The un-deployment process looks exactly the same but it invokes the un-deployment service endpoints to trigger the respective instances' and nodes' deletion.

VII. Evaluation

Over the evaluation, phase a web-based graphical user interface (GUI) has been developed to make the TOSCA2Chef web service interfaces accessible to a test user (Figure 12). The GUI was implemented as a web page that enables uploading TOSCA XML documents and executes the related deployment and un-deployment BPEL processes. The upper section provides an upload feature for TOSCA XML files. The bottom section displays a list of available, already uploaded files and gives means to trigger the deployment and un-deployment process via a button.

The server-side of the web GUI was developed with Java Servlets on an Apache Tomcat server. From the servlets the web service interfaces of the TOSCA2Chef web service are called to put or get TOSCA documents. For the BPEL processes the web service interface of the Apache ODE is called to instantiate a deployment (or un-deployment) process. From

Fig. 11: Deploy BPEL process.

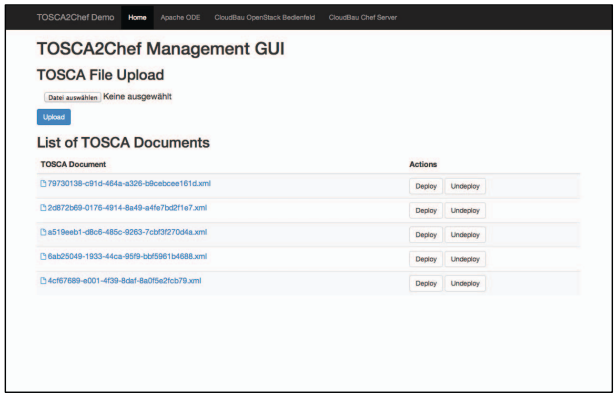

Fig. 12: Evaluation GUI.

the BPEL processes, again, the TOSCA2Chef web service interfaces are called to orchestrate the topology deployment.

The overall deployment and un-deployment processes of the use case through the TOSCA2Chef components developed in this PoC, are demonstrated in the publicly available video: http://www.youtube.com/watch?v=VaPADNi2IAM.

In order to have a reference point of comparison and define some quantitative evaluation metrics (e.g. time to deploy), we implemented the use case scenario through the AWS CloudFormation solution. For those experiments, we emulated a system architecture similar to the original architecture from our experiments with TOSCA2Chef. Unlike the original architecture, the mysql database was not a virtual machine but a hosted service from AWS, i.e., relational database service (RDS), and the load balancer was implementation with AWS's Elastic Load Balancing service (ELB). Aside from these deviations, the architecture of the Cloud Formation experiments as well as with TOSCA2Chef comprise a load balancer, two apache tomcat servers with an example web application, and a Msql database.

Experiments with the prototype give insights regarding the quality of the solution. Within the testbed hardware resources were transparently managed with an OpenStack installation. Additionally, an Opscode Chef Server instance has been put in place to act as a configuration manager of machines within a topology. The OpenStack environment and its endpoints together with the Opscode Chef server make up the testbed.

Two components are hosted on the testbed: (1) an Apache ODE server able to execute BPEL processes, and (2) an Apache Tomcat application server serving the TOSCA2Chef web service, Java servlets and static web content. The Apache ODE server is prepared with two BPEL processes to trigger deployment and un-deployment of the topology described in the use case. The BPEL processes execute web service calls to the Tomcat application server that each trigger the un-/deployment of one of the topology's components. The web service itself executes OpenStack Knife plug-in with a range of parameters on a system level (i.e., Linux shell scripts). A call to the web service is synchronous and a response is sent after a virtual machine instance has been initiated and all software configuration tasks (managed by Opscode Chef) have finished. This blocking behavior is necessary to guarantee a successfully finished deployment but also needed several timeout settings of the Tomcat and ODE server to be touched. Aside from the web service, the web GUI described before is deployed on the Tomcat server.

We have measured the deployment time of 10 deployment runs to calculate an average deployment time for the use case topology. In each test run a deployment of the whole use case topology has been triggered via the BPEL deployment process. The process (see Figure 11) first deploys a MySQL component, then two Tomcat components in parallel, and finally a HAProxy component. The average of 10 successful use case deployment runs is 17 minutes 25 seconds.

The time is an aggregation of the single component deployments, except for the two Tomcat servers which are deployed in parallel. A sequential deployment is necessary to guarantee that Opscode Chef scripts ("recipes") can be applied correctly, particularly, when a configuration recipe's goal is to connect available components. The script will fail if one of the components is not available. Hence, an orchestrating deployment, such as the BPEL processes, and a blocking execution of the deployment tasks is unavoidable.

In contrast, if a configuration management software was able to execute inter-component tasks in a later sequence after the initial configuration has been applied over the whole topology, parts of the deployment could be executed in parallel. Our experiments and time measurements show that the major effort is due to software installations. A parallelization of the software installation phase and a later configuration phase leaves room for a potential decrease in total instantiation time of a Cloud application.

The experimentation with the use case with the CloudFor-

mation solution resulted in an average deployment time of 14 minutes 13 seconds, with a deletion time of 1:30 minutes. The deployment time savings in these experiments may root from the use of hosted services. While the Relation Database Service (RDS) required several minutes to be configured and available (varies from 8-11 minutes), the Elastic Load Balancing (ELB) service showed insignificant delays for configurations (few seconds).

VIII. Summary and Conclusions

The capability of application packaging in Clouds which enables the re-usability and portability of them, is an important precondition to truly realize the often-expressed benefits of virtualized Cloud services. This requires that certain well-defined standards exist, and are generally adopted by the industry. These standards must enable applications to be re-usable, configurable according to certain pre-determined parameters, portable and interoperable between platforms, and enable integration of software and infrastructure modules for PaaS and SaaS.

In this paper we have presented results of an investigation of the current state-of-the-art in a PoC project. In particular, we have selected TOSCA to stress its portability features for a custom Cloud environment. A use case web application has been modeled according to the TOSCA specification. The TOSCA2Chef execution environment has been developed to allow a deployment of the use case application in the Cloud infrastructure test-bed. The experiments prove a successful realization and expose current limitations.

The findings show that TOSCA is still missing essential support to provide a comprehensive solution. Yet, the the contributions made in this paper show that a working solution was found and that TOSCA can be employed to achieve portability to a certain degree. For the future we expect the definition of an execution environment to become part of portability standards such as TOSCA. This allows every Cloud operator to adapt the standard and offer interfaces expected to deploy Cloud applications modeled with TOSCA. Furthermore, the TOSCA specification is not making any suggestions regarding the structure and naming of NodeType entities (resp. RelationshipTypes, ArtifactTypes, etc.), hence, leaving the definition of existing Cloud resources, i.e., virtual machine images and virtual machine configurations, and software configurations of components to every Cloud infrastructure. To foster portability we expect a standard set of entities to be an important issue to be agreed upon in order to enable actual portability between diverse Cloud environments. A support of federated Cloud applications cannot be found in the specification and is subject to future work.

Overall we can comment that the TOSCA specification is an important initiative that is dealing in a holistic way with Cloud application portability and orchestration. The 1st version of the published specification is not mature enough to be directly adopted and integrated in a realistic development environment. The TOSCA Technical Committee is currently working on the second version of the specification which, after having some insight in it, is going to solve many of the inefficiencies of the first version.

References

[1] A. Lenk and F. Pallas, "Cloud standby system and quality model," *International Journal of Cloud Computing (IJCC)*, vol. 1, no. 2, pp. 48–59, 2013.

[2] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, and S. Dustdar, "Moving applications to the cloud: an approach based on application model enrichment," *Int. J. Cooperative Inf. Syst.*, 2011.

[3] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to adapt applications for the cloud environment - challenges and solutions in migrating applications to the cloud," *Computing*, 2013.

[4] S. Strauch, V. Andrikopoulos, T. Bachmann, and F. Leymann, "Migrating application data to the cloud using cloud data patterns," in *CLOSER*, 2013.

[5] T. Binz, G. Breiter, F. Leymann, and T. Spatzier, "Portable cloud services using tosca," *IEEE Internet Computing*, 2012.

[6] O. T. TC. (2012, Nov.) Topology and orchestration specification for cloud applications version 1.0. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/csprd01/TOSCA-v1.0-csprd01.pdf

[7] A.-F. Antonescu, P. Robinson, and T. Braun, "Dynamic topology orchestration for distributed cloud-based applications," in *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, 2012, pp. 116–123.

[8] G. Juve and E. Deelman, "Automating application deployment in infrastructure clouds," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 658–665.

[9] C. Liu, J. E. V. D. Merwe, and et al., "Cloud resource orchestration: A data-centric approach," in *in Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, 2011.

[10] IBM. (2013) Smartcloud orchestrator. [Online]. Available: http://www-03.ibm.com/software/products/us/en/smartcloud-orchestrator/

[11] IDC, IBM, *Orchestration Simplifies and Streamlines Virtual and Cloud Data Center Management*, 2013.

[12] O. C. T. Members. (2012, Aug.) Cloud application management for platforms, version 1.0. [Online]. Available: https://www.oasis-open.org/committees/download.php/47278/CAMP-v1.0.pdf

[13] OpenTOSCA. (2013) Opentosca initiative. [Online]. Available: http://www.iaas.uni-stuttgart.de/OpenTOSCA/

[14] OpenTOSCA. (2013) Opentosca ecosystem. [Online]. Available: http://files.opentosca.de/v1/

[15] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann, "Winery - A Modeling Tool for TOSCA-based Cloud Applications," in *Proceedings of 11th International Conference on Service-Oriented Computing (ICSOC'13)*, Dezember 2013.

[16] OpenTOSCA. (2013) Winery tool. [Online]. Available: http://http://winery.opentosca.org/

[17] I. Amazon Web Services. (2013, Oct.) Aws cloudformation documentation. [Online]. Available: http://aws.amazon.com/en/documentation/cloudformation/

[18] OpenStack. (2013, Oct.) OpenStack Documentation. [Online]. Available: http://docs.openstack.org/

[19] OpenStack. (2013, Oct) Heat - OpenStack Orchestration. [Online]. Available: https://wiki.openstack.org/wiki/Heat

[20] OpsCode Inc. (2013, Oct.) Chef documentation - overview. [Online]. Available: http://docs.opscode.com/chef_overview.html

[21] OpenStack. (2013, Oct.) OpenStack Overview. [Online]. Available: http://www.openstack.org/downloads/openstack-overview-datasheet.pdf

[22] WSBPEL TC. (2013, Oct.) Web services business process execution language version 2.0. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[23] Apache Foundation. (2013, Oct.) Apache ode. [Online]. Available: http://ode.apache.org/

[24] HAProxy. (2013, Sep.) Haproxy tcp/http load balancer. [Online]. Available: http://haproxy.1wt.eu/

[25] The Apache Software Foundation. Apache tomcat 7. http://tomcat.apache.org/. Accessed September 27, 2013.