Cloud Service Orchestration with TOSCA, Chef and Openstack

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
T-Labs (Research & Innovation)
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 project which deals with the deployment and orchestration of Cloud applications through the 1st published version of the TOSCA specification. In our work we investigate the state of the art in the management of Cloud applications and by combining technologies such as Opscode Chef and Openstack we design and develop a framework that allows the management of service topologies on Cloud infrastructures. For the evaluation of the implemented system we orchestrate the un-/deployment of a multi-tier Cloud application use case and in the end we discuss the results and experiences of this activity.**

***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. Furthermore, with growing workload a Cloud service must allow to scale out components of an application as needed. 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 environments are referred to as hybrid Clouds.

A relocation of software applications between substitutable services of competing Cloud providers is not unusual. 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 of the contemporary Cloud applications. While movability and migration have been already investigated in functional level [1][2][3], Cloud service portability in a management level has been only recently explored [4]. To this end, one very interesting activity that brought together various important industrial partners as well as people from academia, is the definition of a Topology and Orchestration Specification for Cloud Applications (TOSCA) [5]. 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 Tehnolgoies, 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 packaging and management of Cloud-enabled applications and their parameters and constraints for deployment to various environments. The activities of this PoC included the state-of-the-art analysis and the selection of the appropriate technologies for carrying out the necessary tasks. In that context, TOSCA has been the technological driver of the PoC implementation as the first standard that has been developed to support the orchestration of Cloud-enabled applications.

In the following section we introduce the state-of-the-art in the field of Cloud application orchestration, and in section III we describe the testbed environment and the fundamental, to this PoC, technologies. In section IV we elaborate on the service orchestration framework that we developed using the TOSCA v1 specification, presenting in details all the phases

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

of this activity, namely the TOSCA2Chef environment, the modeling of the use case with TOSCA and the un-/deployment process.Finally in section V we conclude and discuss the findings of this PoC.

II. State-of-the-art

Cloud orchestration, is the automation of tasks involved with managing and coordinating complex Cloud-enabled software and services. The goal of Cloud orchestration is to automate the configuration, coordination and management of software and software interactions in Cloud environments. This task is sometimes complicated while it involves interconnecting processes running across heterogeneous systems in multiple locations, usually with proprietary interfaces.

Lately there have been several initiatives and efforts so much in research [6] [7] [8] [4] but also in industry [9][10] dealing with Cloud application orchestration 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 high level of adoption. There are also others that receive the support from bigger part of industry and they 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)[11] 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. All resources listed, besides the platform, can exist in an instantiated and in a deployed form (templates). 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.

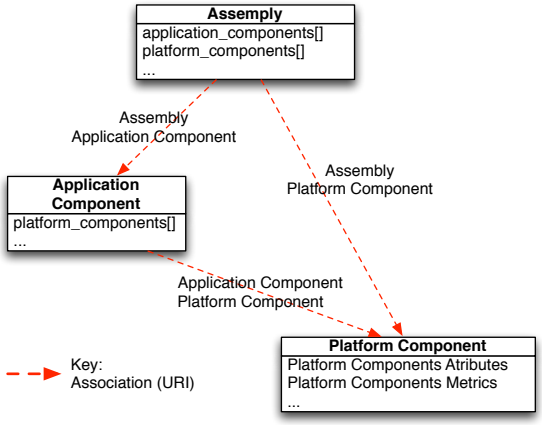

Fig. 1: CAMP Ressource Relationships - Assemblies and Components. Source: [11]

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) [5] 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

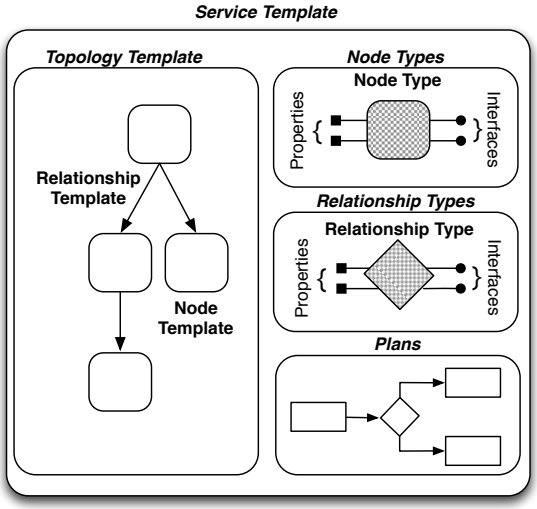

Fig. 2: Structural Elements of a Service Template and their Relationships. Source: [5]

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) were the first open source available software published through the OpenTOSCA initiative [12][13]. Specifically a modeling tool called Winery [14][15] was published, through which one can define a Cloud application topology using a web GUI, and 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 which was at that point only the first version of the TOSCA specification documentation.

C. AWS CloudFormation

In contrast to the previous general specifications, Amazon Webservices (AWS) CloudFormation [16] 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 [17] is one of the most popular Cloud middleware software stacks, and thus they suggest their own orchestration program: Heat [18], 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 [17] based infrastructure was leveraged, using the Opscode Chef [19] platform to perform, manage and automate virtual machine (VM) deployments and the software packages installations over the infrastructure. The management of both systems is achieved through the Knife client. 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 TOSCA2Chef runtime environment.

A. OpenSTACK

For the infrastructure provisioning on the testbed, we relied on OpenStack [17]. 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 [20], OpenStack consists of three main projects: (a) OpenStack Compute, a scalable compute provisioning engine, (b) OpenStack Object Storage, a fully distributed object store, and (c) OpenStack Imaging Service, an image registry and delivery service.

B. Opscode Chef

Configuration management was handled using Opscode Chef [19]. 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 choose 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 [21]. As a BPEL-engine, allowing us to execute these plans, the open source BPEL-engine Apache Orchestration Director Engine (ODE) was used [22].

IV. TOSCA2Chef service orchestration framework

In order to perform in an end-to-end fashion this PoC, and considering the limited available resources and TOSCA related tools during the period of development, in this project we had to implement a functional TOSCA-enabled runtime environment, we had to design node hierarchies and relations to model our use case and also define un-/deployment processes to put the topology execution in an order. As it is presented in the following subsections, the necessary developments of this PoC were split and grouped into three tasks: (a) the development of the TOSCA2Chef runtime environment, (b) the modeling of the use case using TOSCA, and (c) the un-/deployment process definition.

A. *TOSCA2Chef runtime environment*

The core entity of the proposed architecture is the TOSCA2Chef runtime environment: a set of components and services that parses a TOSCA document, extracts the information regarding the described application and triggers the necessary commands execution towards the Knife client.

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 unemployment) TOSCA plans. The TOSCA plans are defined as process models, i.e. a workflow of one or more steps. TOSCA specification relies on existing languages like BPMN or BPEL in order to capture such plans. Therefore, in our experimentation, the plan execution operation is realized using the Apache ODE BPEL engine [REF]. On the other hand, the parsing of the TOSCA document and execution of the necessary commands towards the test-bed is being managed through the TOSCA Container web service (Figure 3).

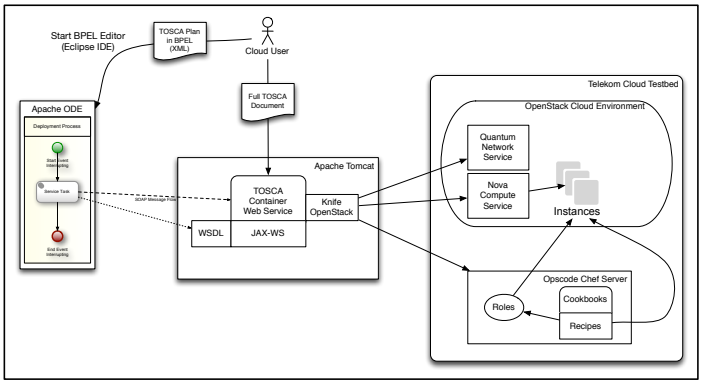

Fig. 3: High level architecture of the TOSCA2Chef execution environment.

For the effective management of the TOSCA topology descriptions we had to introduce certain interfaces facilitating the file management and plan execution. The API that have been designed and exposed by the respective service is presented in Figure 4.

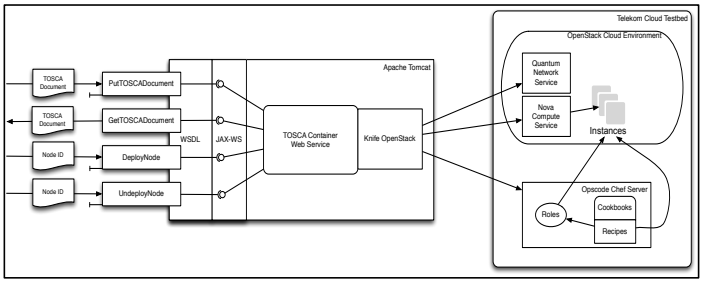

Fig. 4: TOSCA Conatiner API.

The implementation of the TOSCA container is required to parse a TOSCA XML document that describes an application topology. Furthermore, the web service acts as a processor of (un-)deployment requests for TOSCA XML documents. To transform a parsed TOSCA XML document into running VMs in the testbed, an intermediary data model was required that reflects a TOSCA topology in terms of Chef and Openstack. The intermediary model builds the basis to deploy - or undeploy - VMs via the Knife-Openstack client. Given that a TOSCA model derives from TOSCA XML document, a transformation logic can derive an intermediary data model.

In order to enable the deployment of the described application to a specific test-bed environment, an intermediary data model (Figure 4) 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 5 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. Also, the plan execution operation must be adapted to be able to base its execution logic on the new model.

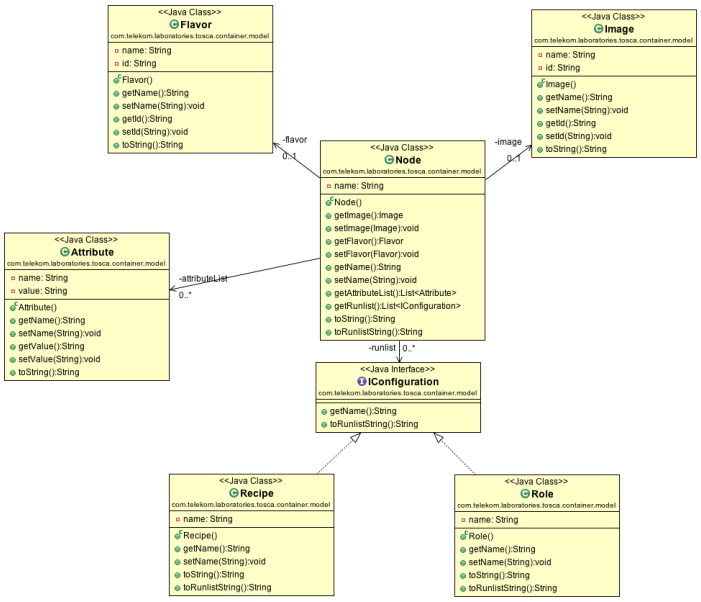

Fig. 5: Intermediate, domain specific data model.

Since no tools were available for parsing a TOSCA document (at the moment of the PoC implementation), a library for the XPath XML query language was employed to replace a missing TOSCA model. Information from a TOSCA XML file is extracted with a set of XPath queries and stored into an Openstack- and Chef-compatible intermediary data model. An execution engine based on the intermediary data model orchestrates the deployments.

B. *Modeling the use case with TOSCA*

For the realization of this proof of concept project a specific multi-tier application service topology had do be designed as

the use case through which we would evaluate the related technologies. The selected application is a basic 3-tier web application, comprising a Load Balancer component implemented through HAProxy open source software package [23] and two application servers implemented through Apache Tomcat [24], 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. The graphical representation of the topology is presented in Figure 6.

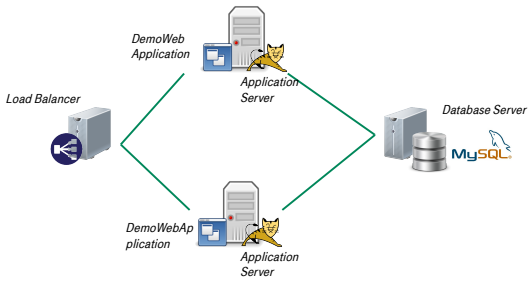

Fig. 6: The selected 3-tier web application use case.

For the description of the use case topology following the TOSCA specification, a series of NodeTypes, RelationshipTypes and ArtifactTypes have been defined in order to capture all the entities of the application and their interrelation. While those types are actually re-usable definitions, they could populate a pool of resources that could be used by application developers for other service topology descriptions. The NodeTypes definitions of the use case application are presented in Figure 7.

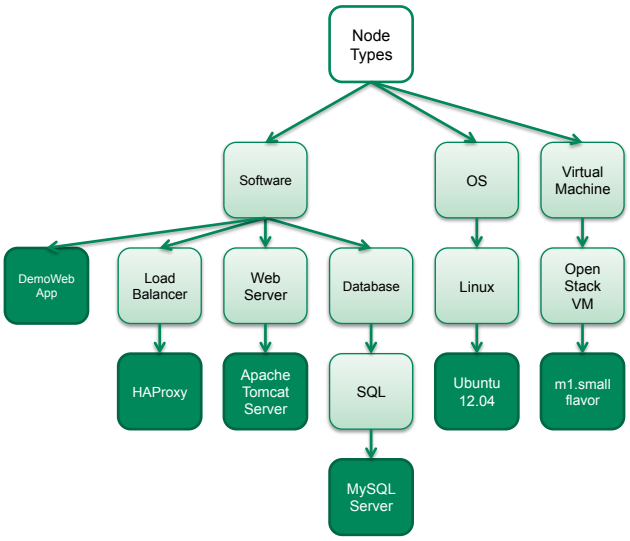

Fig. 7: Nodetypes definitions.

The TOSCA specification supports an inheritance functionality (DerivedFrom tag), which allows us to design our application 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 7) 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 8 we visually present the definitions of the NodeTypeImplementations and DeploymentArtifacts for our use case application.

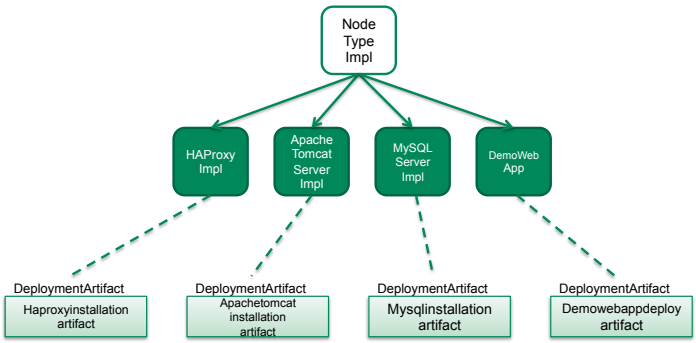

Fig. 8: NodeTypeImplementation diagram and Deploymentartifact definitions.

The following part of example XML code describes the implementation for the NodeType HAProxy. We use the RequiredContainerFeature functionality 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 *haproxyinstallationtemplate* which includes the implementation scripts or commands. As can been seen in the following lines of code, the artifact include the declarations of the roles and/or recipes that are 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>
```

A functional topology was defined, capturing the relationships between the nodes as well as the multiple instances of some NodeTypes. In Figure 9 we can see the Topology Template of the use case application depicting 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 that we used to demonstrate the operation is hosted on an Apache Server.

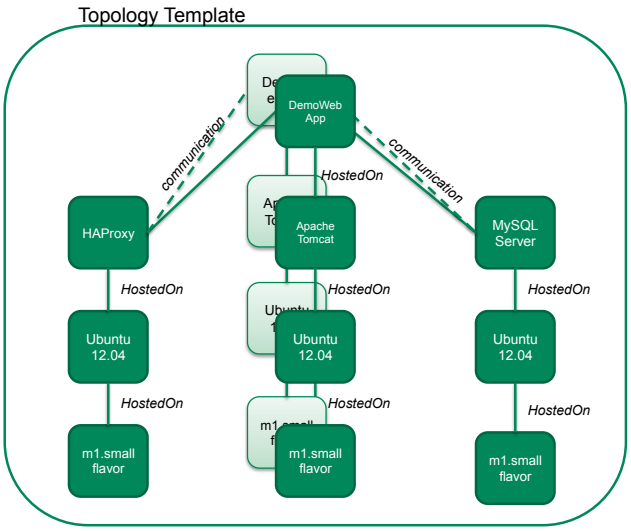

Fig. 9: Use case Service Topology design.

In the use case application the HAProxy component is balancing the load between the two DemoWebApp deployments, which in their turn communicate with the MySQL server to retrieve the relevant data. Based on the general HostedOn and Communication relationships, 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 10 we present all the relationship types and their hierarchy. In the same way that nodes are instantiated (through artifacts), the relationships can also be configured. In that context, Implementation Artifacts can be defined to capture certain parameters of a relationship between two nodes.

C. Use case un-/deployment process

After the long lasting state-of-the-art analysis, design phase, and development of the orchestration framework, this PoC project dealt with the evaluation of the implemented use case scenario. During that phase 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

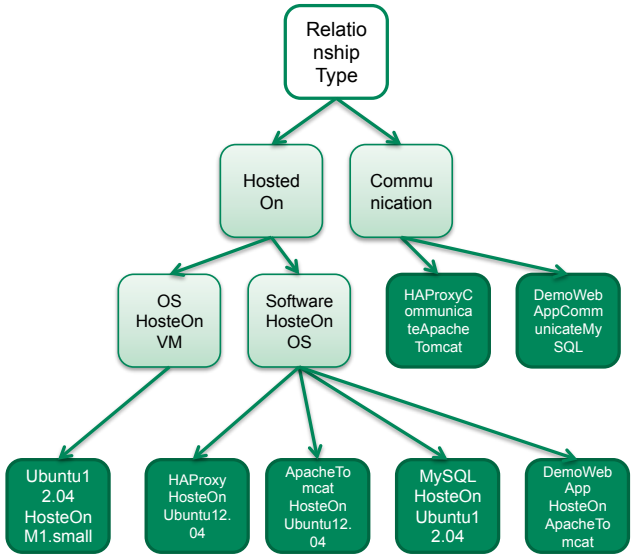

Fig. 10: Relationship types definitions.

the un-deployment service endpoints to trigger the respective instances and nodes deletion.

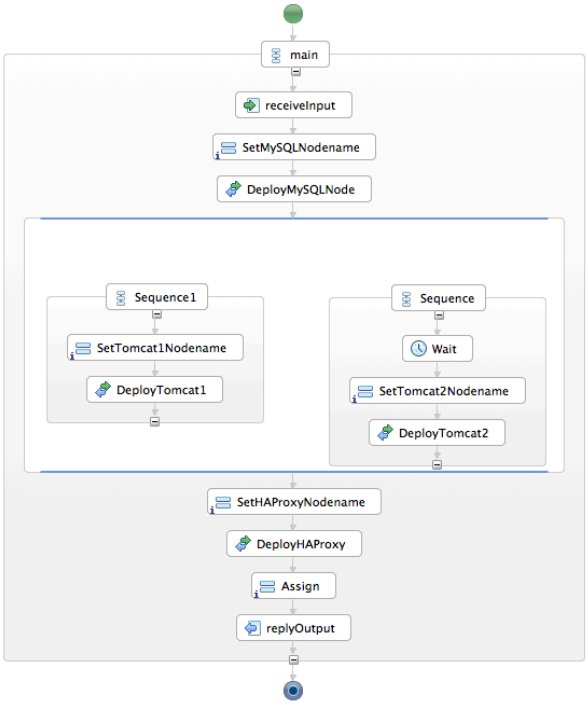

Fig. 11: Deploy BPEL process.

Over this 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 pro-

vides 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 a 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 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 that were developed in this PoC, are demonstrated in the publicly available video: http://www.youtube.com/watch?v=VaPADNi2IAM.

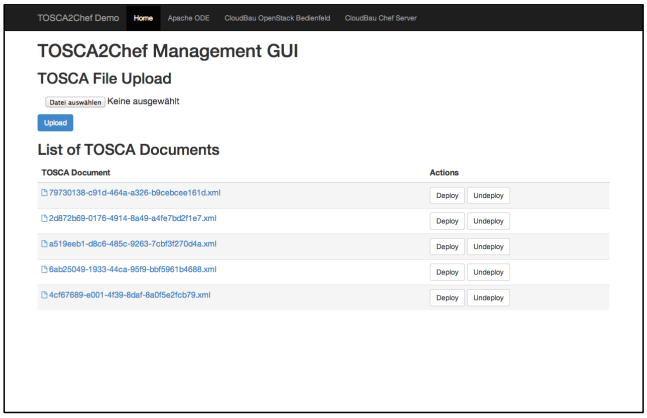

Fig. 12: Evaluation GUI.

Experiments with the prototype give insights regarding the quality of the solution. To conduct the experiments we used a testbed operated by Telekom Laboratories. 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.

On the testbed run 2 components: (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 2 BPEL processes to trigger deployment and undeployment 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 a small web front-end is deployed on the Tomcat server. The front-end gives means to upload a new TOSCA document and trigger the deployment or un-deployment of an already uploaded TOSCA document.

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, the deployment part can be parallelized. 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 would show a tremendous decrease in total deployment time.

V. 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. In addition, common deployment processes need to be established as part of an overall Service Orchestration.

Service Orchestration has to address a number of dimensions including: (a) Commercial orchestration (contract and financial aspects and pre-requirements), (b) Technical Service Orchestration (actual system deployments), (c) Service Orchestration (configuration of systems according to SLAs with support teams invocation, reporting etc.), and (d) Consumer interface management (updating status etc. in the Portal UI GUI, and via the API).

Previous PoC projects [25][26] have proven that the above dimensions can be orchestrated up to the VM level (and in some cases even the Operating System level), based on the industry adoption of standards such as the DMTFs OVF. The capability for Service Orchestration at the PaaS and SaaS levels, leveraging industry standard frameworks and templates is still extremely limited. In our proof-of-concept activity we tried to investigate the next layers up into the Application stack - to support PaaS and SaaS by exploiting the industrially-endorsed TOSCA standard in a realistic use case deployment scenario.

A. PoC findings

During the development of this PoC we gained valuable experience related with service orchestration approaches using TOSCA. Those findings can be summarized in the following points:

- TOSCA Specification v1.0 introduces a new set of entities and concepts used in the definition of service topologies. For the Cloud service specification the exact types and hierarchies of types must be defined.
- Without a realistic pool or library of resources/TOSCA-types for the nodes, relationships and artifacts the overhead for the application development is relatively big: TOSCA specification in the current form and with the current available resources is abstract and therefore deriving the required Cloud resources and configuration management actions becomes a challenging task.
- A domain specific extension to TOSCA helps to define details needed to derive a system configuration. This can be achieved through the introduction of data model, related with current Cloud deployment architectures and supported by several pre-defined TOSCA Node and Relationship Types.
- The functionality and concepts for the support of multiple NodeType implementations in the same topology description and the effective, automated orchestration of such use case is under-specified. With the current specification multiple NodeTypeImplementations can be introduced, but the selection logic lies ultimately on the parser / orchestrator component.
- There are no typical models or recommended ways yet to apply an actual deployment process by means of Plans. The part of service orchestration with TOSCA that is related with the plans execution is not satisfactory documented.
- There is no official TOSCA Container implementation available that supports the execution of Plans, and little documentation describing communication between a BPMS executing a Plan, and the container. No suggested API is available, thus the implementation of such component is highly related with the domain of the application or the underlying infrastructure (e.g. in our PoC Openstack and Chef).

Overall we could comment that the TOSCA specification is an important initiative that is dealing in a holistic way with Cloud application portability and orchestration. The v1 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] 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.

[2] 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.

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

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

[5] 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

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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

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

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

[14] 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.

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

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

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

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

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

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

[21] 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

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

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

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

[25] Open Data Center Alliance. (2012) Virtual machine (vm) interoperability in a hybrid cloud environment rev. 1.1. [Online]. Available: http://www.opendatacenteralliance.org/docs/VM_Interoperability_Rev_1.1_b.pdf

[26] Open Data Center Alliance. (2013) Implementing the open data center alliance virtual machine interoperability usage model. [Online]. Available: http://www.opendatacenteralliance.org/docs/VM_Interop_PoC_White_Paper.pdf