OPEN DATA CENTER ALLIANCE℠: SERVICE ORCHESTRATION WITH TOSCA WHITE PAPER

TABLE OF CONTENTS

CONTRIBUTORS

The following individuals contributed to the content of this document:

Dr. Gregory Katsaros,
FZI Forschungszentrum Informatik

Alexander Lenk,
FZI Forschungszentrum Informatik

Michael Menzel,
FZI Forschungszentrum Informatik

Ryan Skipp,
T-Systems International

Dr. Jannis Rake-Revelant,
Telekom Innovation Laboratories

LEGAL NOTICE

OPEN DATA CENTER ALLIANCE℠ USAGE MODEL: SERVICE ORCHESTRATION WITH TOSCA WHITE PAPER

EXECUTIVE SUMMARY

Enterprise IT operators derive three main benefits from the adoption of cloud infrastructure technology:

1. **Performance–Near real-time availability of cloud services**
2. **Cost–Cloud services may be facilitated to gain elastic software behavior, billed per use (scalability with metered resource usage)**
3. **Capabilities–Provision service offerings from a market with multiple competing providers and consumers who share the cost of development and lifecycle evolution result in constant service evolution and competitiveness, with new functions**

Cloud infrastructure services allow the operation of software applications on virtual machines (VM), managed in virtualized environments. Virtualization technology helps enable automated creation of VMs and exposes 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 help to enable deployments of a software application over multiple VMs. Furthermore, with a growing workload, a cloud service must help enable the consumer to scale application service components out as needed. This guarantees immediate alignment with business requirements. Additionally, software applications might be operated on VMs of various cloud providers. VMs utilized in such a setup may be part of a private cloud and/or a public cloud. Unlike public clouds, private clouds are operated and maintained in a company's own data center/s. Deployments incorporating a mix of public and private cloud environments are called 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 also 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.

A team led by T-Systems International, Telekom Innovation Laboratories, and the FZI Forschungszentrum Informatik research team from the University of Karlsruhe carried out a proof of concept (PoC) project to test packaging of an application and its parameters for deployment to various cloud environments. Experiments were isolated to a single application packaging approach and cloud environment. With the Topology and Orchestration Specification for Cloud Applications (TOSCA) framework, an approach that claims portability has been chosen. One of the major challenges within the PoC is to transform definitions of packaged application components to an actual application operated on cloud resources. A system to unpack and deploy a TOSCA definition to cloud services is not available yet. Furthermore, it is necessary to follow the Service Orchestration Master Usage Model (MUM) when developing a test environment. A system that unpackages and deploys an application must be capable of orchestrating different services.

To address the MUM and construct an environment to test the TOSCA framework, existing software products designed to realize a private cloud and support application deployments were surveyed as part of the project underpinning this PoC. The survey covered most publicly available included software products employed to operate cloud infrastructures commonly found in private clouds. A set of appropriate experiments was defined to test the interplay of the multiple software products and TOSCA regarding the MUM. The results presented in this document demonstrate how effectively software applications could be defined, packaged, and deployed. The findings demonstrate the maturity of cloud technologies and current standard specifications to support the MUM.

The results represent a minimum baseline for expectations regarding the current "state of the art" for cloud technologies. We expect technologies for portable packaging of software applications to advance continuously. Tests conducted in the near future are expected to show significant improvements over current findings.

THE TOSCA-BASED APPLICATION PACKAGING PROJECT

The goal for this project was to devise and implement testing methods, and conduct a basic set of functional tests for the Service Orchestration of a simple application in a cloud environment using the TOSCA specification framework[1].

Three prominent possible frameworks are identified in this area: (1) CAMP, (2) TOSCA, and (3) Heat DSL. In an initial step of the PoC a survey to compare all three approaches was conducted. The following results and key characteristics were identified through the survey:

1. OASIS CAMP (Cloud Application Management Platform[2])
   - Key concepts include ComponentTemplates, which are pieces available in the platform
   - PlatformComponentTemplates are supplied by the platform; for example, a database or a web server cluster
2. OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications[3])
   - Key concepts include modeling topologies
   - Service topologies are described using the TOSCA “meta-model”:
     - Nodes and relationships represent
       - Components of an application or service
       - Logical relationships between components and their properties.
       - Example nodes include: Infrastructure nodes such as compute, network, storage, or higher level node types such as OS, VM, DB, web server, etc.
3. Heat DSL (Domain-Specific Language) for the API for deploying applications[4]
   - CloudFormation compatibility API and templates, Heat also sports an OpenStack-native rest API, native representations of OpenStack resources, and a richer template format.
   - Key concepts include a vocabulary:
     - Resources are items natively available from OpenStack
     - Stacks are composites of these resources and/or other stacks
   - Proprietary but isomorphic to a subset of CAMP

For the PoC, the OASIS TOSCA specification, reported to be significantly open and appropriately generic, was considered for further experiments. TOSCA supports both enterprise and open-source concepts adequately. Another feature of TOSCA is its extensibility. Consequently, it is able to map application component definitions, such as a web server, directly to arbitrary cloud infrastructure service implementation, i.e., an OpenStack VM image and VM configuration (a flavor in terms of OpenStack). In addition, TOSCA has been supported, adopted, and researched by several academic or industrial partners, such as the University of Stuttgart (OpenTOSCA[5]) and IBM (SmartCloud Orchestrator*[6]).

Ideally, cloud subscribers would like to be able to select any cloud provider on the basis of service cost, performance, and capabilities. They also want to link, as needed, private clouds made up of dedicated services with public clouds that consist of shared services. In order to make this feasible for the cloud consumer, it is necessary to be able to package and deploy applications to private and public clouds. A packaging format is able to foster interoperability and portability in virtual environments, preferably leveraging defined industry standards. Furthermore, to fully provide interoperability between cloud services, a packaging format must be capable of quantifying, packaging, cross-mapping and associating defined resources of a of cloud-based application. In parallel, an execution environment for packaged cloud applications must facilitate an automated deployment. It must apply and transfer defined resources to their destination with a reasonable degree of reliability.

Reliability and reproducibility of an activity such as an application deployment to multiple cloud services is based on well-integrated defined standards, specifications, frameworks, scenarios, and processes. All cloud provider participants, such as a private cloud operator and public cloud provider, must therefore adhere to a “common” standard. Thereby, a federated cloud environment with a single application packaging standard emerges. If this is achievable, then cost efficiency is possible for operations (reduced duplication of work and integration effort), and true federated deployments may occur with minimized risk of restrictions appearing later on.

[1] www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[2] www.oasis-open.org/committees/camp
[3] OASIS (Organization for the Advancement of Structured Information Standards) is a non-profit consortium that drives the development, convergence, and adoption of open standards for the global information society (see www.oasis-open.org) including the Topology and Orchestration Specification for Cloud Applications (TOSCA), for which v1 has been publicly released (www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca).
[4] www.openstack.org/summit/san-diego-2012/openstack-summit-sessions/presentation/heat-a-template-based-orchestration-engine-for-openstack
[5] www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php
[6] http://public.dhe.ibm.com/common/ssi/ecm/en/til14066usen/TIL14066USEN.PDF

The participants and members in the ODCA, with practitioners representing most of the aforementioned scenarios above, believe that this need exists in their organizations. The major goal of the PoC is to determine if such a packaging format and execution environment could be built, and whether technology and standards adoption is yet at a level to help enable the "cloud vision" of application interoperability through use of standard packaging formats. This implies that a PoC implementation is able to move applications between cloud services, and to cross-map and deploy between cloud providers if necessary. Additionally, the goal of the PoC is to prove that it is capable of helping prevent any vendor lock-in; for instance, no relocation options once an application has been deployed (i.e., avoid creating "future legacy environments").

PoC participants agreed to focus on creating sample packages with TOSCA to determine the necessary concepts and characteristics of a package structure. Given the understanding of TOSCA concepts, a use case of a multi-tier application was defined and packaged. A cloud testbed provided the infrastructure to deploy applications onto. Within the testbed the PoC implementation disclosed an interface to deploy application packages. The use case package was then deployed and un-deployed using the PoC implementation of an execution environment. In doing so, the PoC implementation was expected to orchestrate cloud services during a deployment according to the ODCA Service Orchestration Master Usage Model[7].

The PoC project included the following component tasks:

- Definition of target nodes for deployment of a use case application
- Assembly of a package according to the TOSCA specification that represents a multi-tier application deployment
- Development of the runtime environment that could manage the previous package
- Orchestration of the package deployment on a cloud testbed environment
- Summary of the findings regarding the applicability of the use case in larger production environments

Overall, it was demonstrated that with the right preparation and interpretations, applications could be successfully packaged and deployed using the TOSCA framework between various platforms. However, the TOSCA framework restricts its support to an abstract package definition format. There are a variety of options to define topologies of application components within a package. The definitions in a package and the implementation of an execution environment must agree to the same concepts and structures. Given this abstraction and openness, an execution environment implementation must dictate a proprietary packaging format. A standard supported by multiple participants is, therefore, required to help enable portability for TOSCA-based packages.

During the development of this PoC it was also identified that there were no publicly available tools in a formally released version that adopt and fully align to TOSCA. The first version of the OpenTOSCA initiative was released on September 11, 2013[8]. Therefore, the presented tools (namely, Winery, OpenTOSCA Container) were not available to the members participating in this PoC. Consequently, the portability of packages is somewhat proprietary to every implementation. The definition of a standard and an alliance between cloud providers accepting the standard remains a future task.

[7] "*Open Data Center Alliance Usage: Service Orchestration Master Usage Model.*" See www.opendatacenteralliance.org/library

[8] http://files.opentosca.de/v1

IMPLEMENTATION OF THE USE CASE WITH TOSCA

For the realization of this PoC, the team had to specify and establish an appropriate use case and define the various nodes for it up front. The selected application was a basic web application, consisting of a load balancer (HAProxy*), two application servers (Apache Tomcat*), and a database server (MySQL*). The graphical representation of the topology is represented in Figure 1.

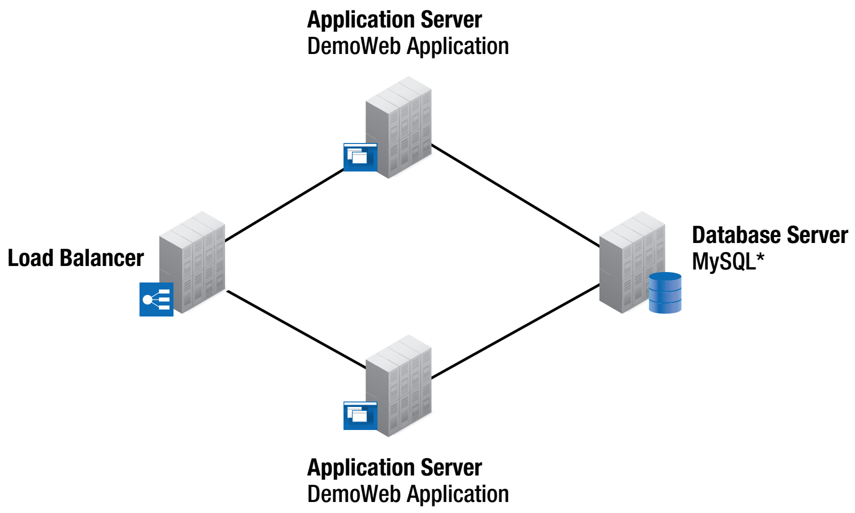


Figure 1: Use Case Application Topology

A series of TOSCA *NodeTypes* were defined to separate the introduced elements so that they could be associated in re-usable definitions with each other. This resulted in the following structure represented in Figure 2.

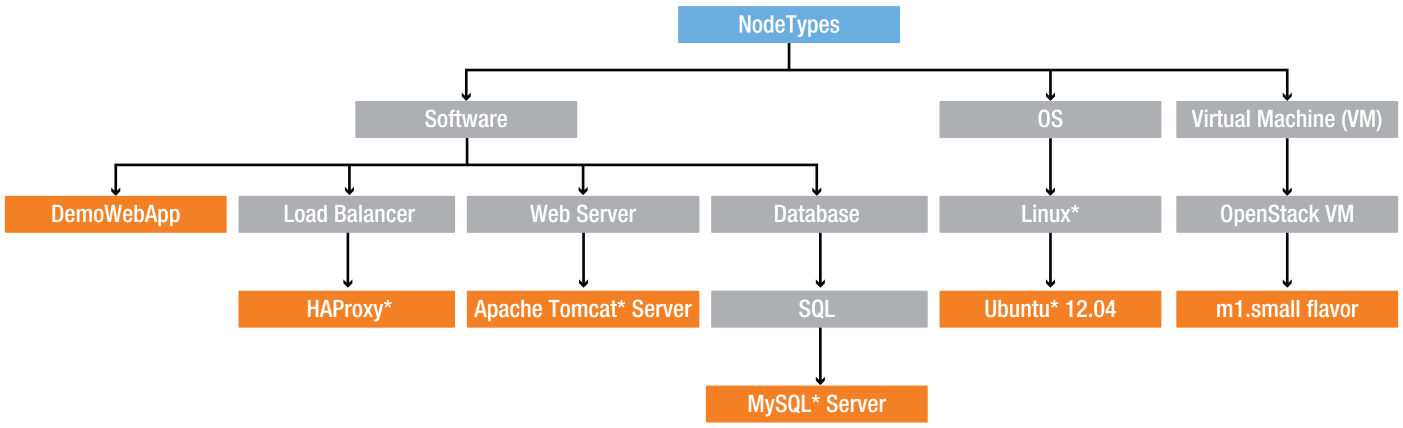


Figure 2: TOSCA NodeTypes Hierarchy

The TOSCA specification supports an inheritance functionality (*DerivedFrom* tag), which allows the developer to design the application and define several levels and types of nodes. In that context, the team distinguished the nodes according to three basic categories: *software, operating system* and *virtual machine*. In addition, the team defined different abstractions for each category such as *database, MySQL* or *Linux*,* etc. The gray *NodeTypes* are the final types that are also implemented (*NodeTypeImplementation* element) and compose the defined use case for application deployment. Every *NodeTypeImplementation* declares a *DeploymentArtifact,* which is the one that actually includes the scripts (in this case, Chef Roles and Recipe names) in order for the node to be instantiated during the deployment. Figure 3 visually presents the definitions of the *NodeTypeImplementations* and *DeploymentArtifacts* for the specific use case application.

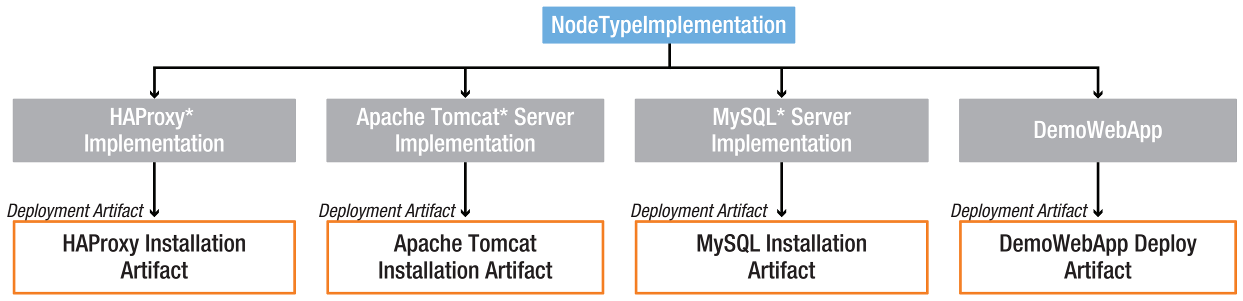

NodeTypeImplementation
HAProxy* Implementation
Apache Tomcat* Server Implementation
MySQL* Server Implementation
DemoWebApp
Deployment Artifact
HAProxy Installation Artifact
Apache Tomcat Installation Artifact
MySQL Installation Artifact
DemoWebApp Deploy Artifact

Figure 3: NodeTypesImplementations for the Use Case Topology

A functional topology was defined, capturing the relationships between the nodes as well as the multiple instances of some *NodeTypes*. Figure 4 shows 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 VM instance. The *DemoWebApplication* that was used to demonstrate the operation was hosted on an Apache Tomcat Server.

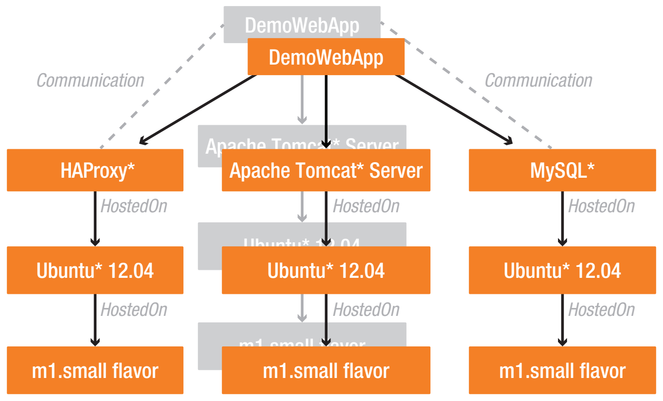


Figure 4: TOSCA Use Case Topology Template

In the use case of the application, the HAProxy component balanced the load between the two *DemoWebApplication* deployments, which in turn communicated with the MySQL server to retrieve the relevant data.

Based on the general *HostedOn* and *Communication* relationships, the team 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. Figure 5 presents all the relationship types and their hierarchies. Similar to the way that nodes are instantiated through artifacts, the relationships are also realized. In that context, *Implementation Artifacts* could be defined to capture certain parameters of a relationship between two nodes. For the realization of the defined use case, the team defined the *Implementation Artifacts* presented in Figure 6.

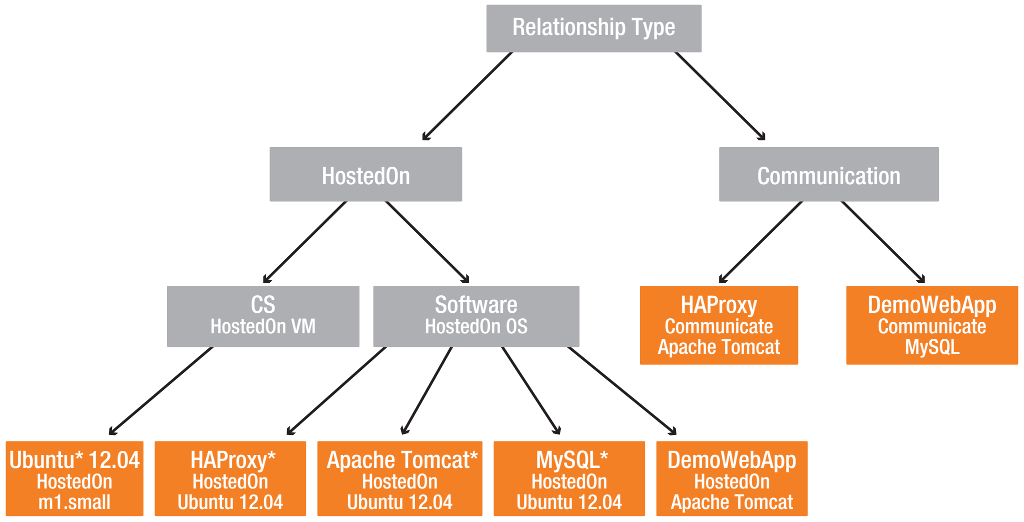

Relationship Type
HostedOn
Communication
CS HostedOn VM
Software HostedOn OS
HAProxy Communicate Apache Tomcat
DemoWebApp Communicate MySQL
Ubuntu* 12.04 HostedOn m1.small
HAProxy* HostedOn Ubuntu 12.04
Apache Tomcat* HostedOn Ubuntu 12.04
MySQL* HostedOn Ubuntu 12.04
DemoWebApp HostedOn Apache Tomcat

Figure 5: Relationship Types Diagram

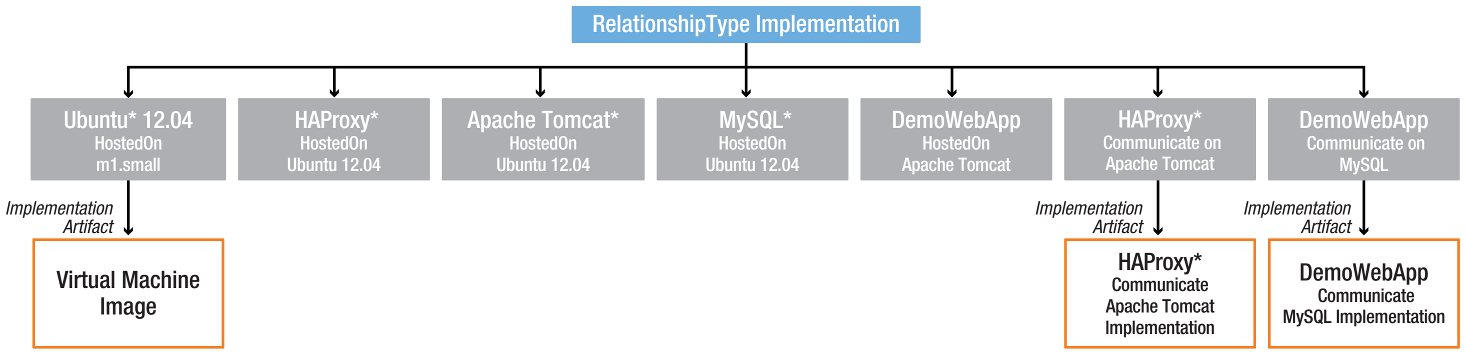

RelationshipType Implementation
Ubuntu* 12.04 HostedOn m1.small
HAProxy* HostedOn Ubuntu 12.04
Apache Tomcat* HostedOn Ubuntu 12.04
MySQL* HostedOn Ubuntu 12.04
DemoWebApp HostedOn Apache Tomcat
HAProxy* Communicate on Apache Tomcat
DemoWebApp Communicate on MySQL
Implementation Artifact
Virtual Machine Image
Implementation Artifact
HAProxy* Communicate Apache Tomcat Implementation
Implementation Artifact
DemoWebApp Communicate MySQL Implementation

Figure 6: Implementation Artifacts Linked to Relationship Type Implementations

The entire TOSCA XML file representing the above mentioned use case is provided in the Appendix at the end of this document.

TESTBED OVERVIEW AND HIGH-LEVEL ARCHITECTURE

The testbed used in this PoC was hosted in the Telekom End-user Integration Center Test Lab Cloud in an isolated network partition. Existing OpenStack[9]-based infrastructure was leveraged using the Opscode Chef[10] platform to perform, manage, and automate VM deployments and the software packages installations over the infrastructure. Chef configures existing VMs through the definition and usage of *cookbooks, roles,* and *recipes* in Chef terminology. The management of both systems is being achieved through the Knife client. The interplay of OpenStack and Chef is a frequent combination. Therefore, the community of developers enhancing Chef has developed an extension to the Knife client. The extension integrates Chef configurations of VMs with the OpenStack APIs to create and manage VMs. The extended Knife-OpenStack client provides the means to instantiate and configure VMs with a single interface.

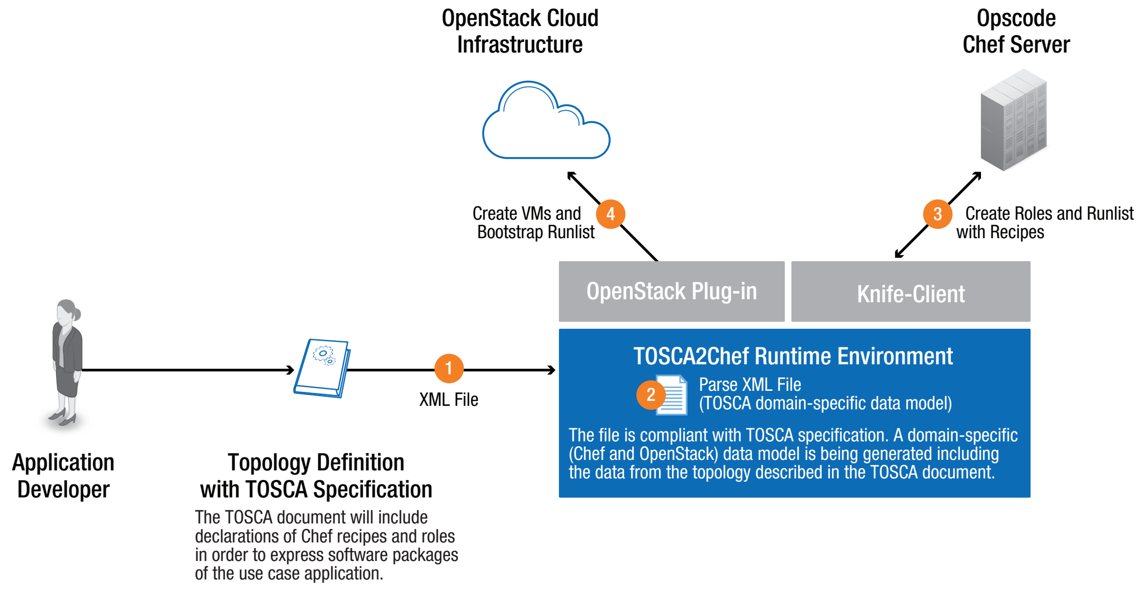

OpenStack Cloud Infrastructure
Opscode Chef Server
Create VMs and Bootstrap Runlist
4
3
Create Roles and Runlist with Recipes
OpenStack Plug-in
Knife-Client
TOSCA2Chef Runtime Environment
2
Parse XML File (TOSCA domain-specific data model)
The file is compliant with TOSCA specification. A domain-specific (Chef and OpenStack) data model is being generated including the data from the topology described in the TOSCA document.
1
XML File
Application Developer
Topology Definition with TOSCA Specification
The TOSCA document will include declarations of Chef recipes and roles in order to express software packages of the use case application.

Figure 7: High-Level Architecture of the PoC Implementation

Figure 7 presents the high-level architecture of the PoC implementation. 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 the described 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 Business Process Modeling Notation (BPMN) or BPEL in order to capture such plans. Therefore, in this PoC, the plan execution operation is realized using the *Apache ODE* BPEL engine*[11]. On the other hand, the parsing of the TOSCA document and execution of the necessary commands towards the testbed is being managed through the TOSCA Container web service (Figure 8).

[9] www.openstack.org
[10] www.opscode.com/chef
[11] http://ode.apache.org

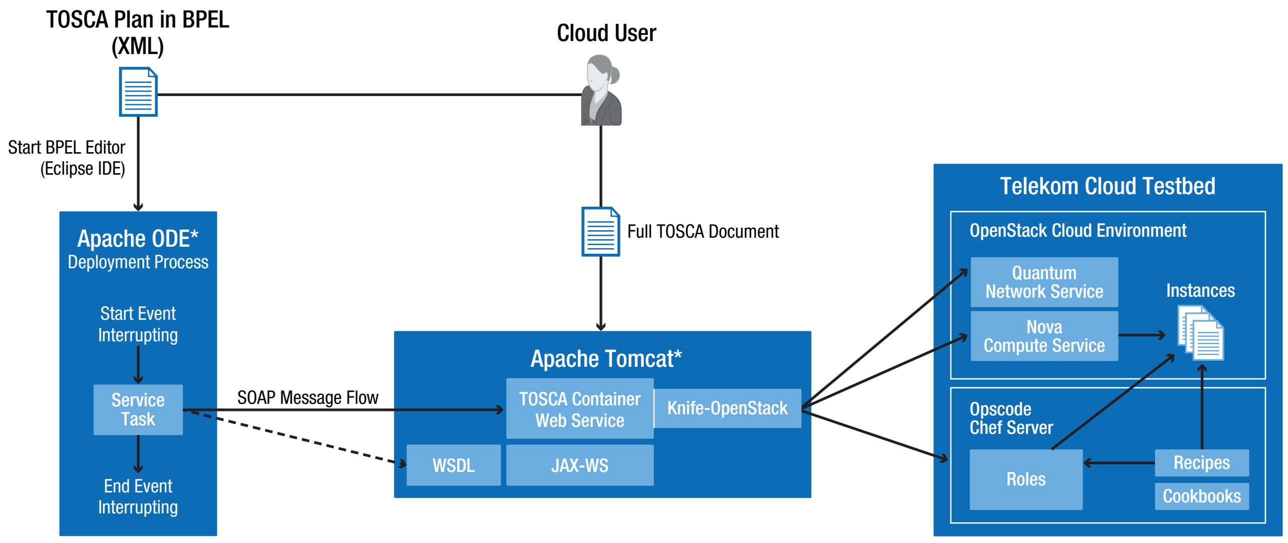


Figure 8: PoC Detailed Architecture

The TOSCA Container web service that was implemented in this PoC exposes a certain API in order to interact with the Business Process Management System (BPMS) engine and manage a TOSCA XML file. The specific interfaces exposed can be seen in Figure 9.

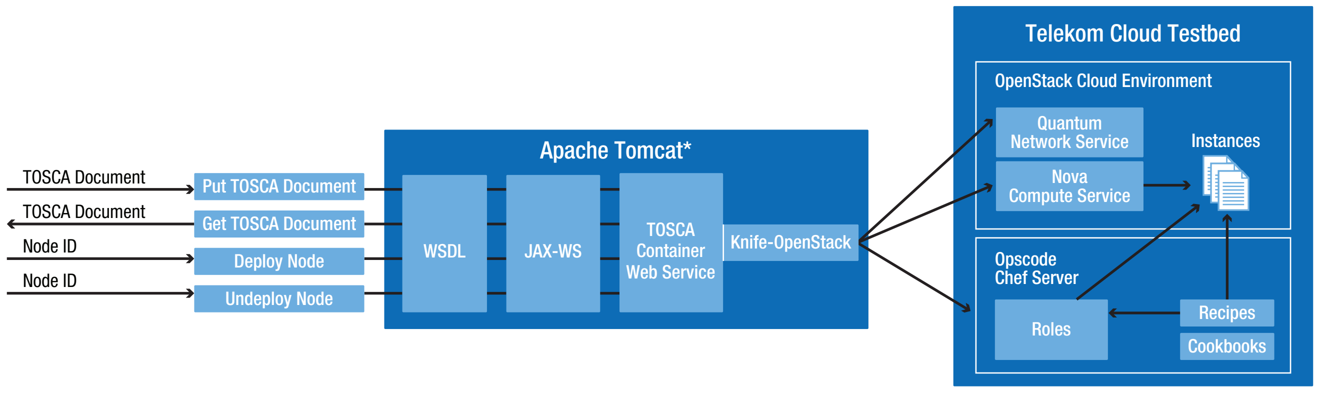


Figure 9: TOSCA Container Web Service APIs

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 un-deploy – VMs via the Knife-OpenStack client. Given a TOSCA model derived from TOSCA XML document, transformation logic could derive the intermediary data model.

In order to help enable the deployment of the described application to a specific testbed environment, the intermediary data model must include some domain specific entities that are relevant with the technologies used in the Cloud testbed. 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 and captured the 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 user the new data-model, as well as the plan execution operation, to be able to read form the new model.

Since no tools were available for parsing a TOSCA document, 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. By accessing the intermediary model it derives calls to a Knife-OpenStack client. The overall process is presented in Figure 11.

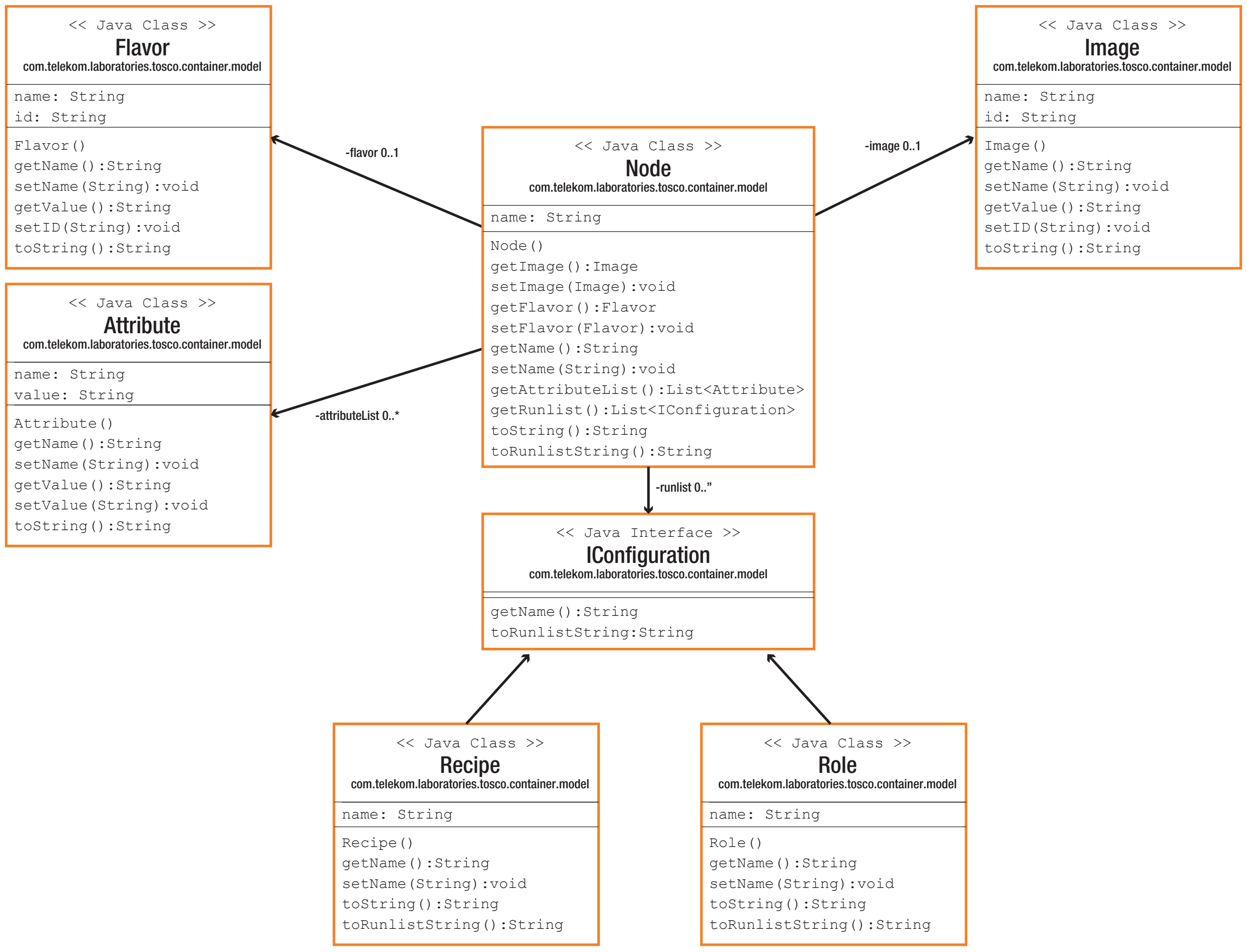


Figure 10: OpenStack and Opscode Chef Data Model

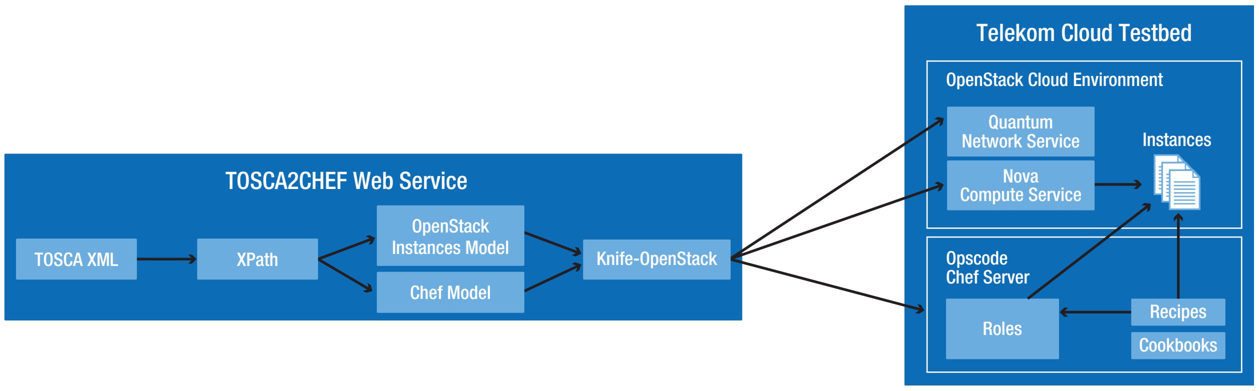


Figure 11: TOSCA2Chef Internal Execution Process

PROOF OF CONCEPT EVALUATION

The PoC phase took approximately four months. It was staffed with a team of specialists from T-Systems International, Telekom Innovation Laboratories, and FZI Forschungszentrum Informatik at the Karlsruhe Institute of Technology. The activities during the PoC development and evaluation were performed based on the following roadmap in Figure 12.

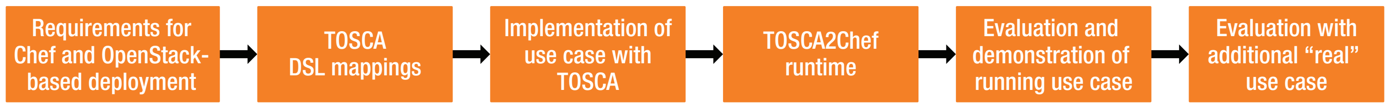

Requirements for Chef and OpenStack-based deployment
TOSCA DSL mappings
Implementation of use case with TOSCA
TOSCA2Chef runtime
Evaluation and demonstration of running use case
Evaluation with additional "real" use case

Figure 12: PoC Development Plan

According to the roadmap, the requirements for the deployments to the provided testbed were initially defined. This step involved configuring and testing the testbed regarding deployments with OpenStack interfaces, Chef configurations, and the Knife-OpenStack client. Subsequent steps benefit from the insights gained during tests with the testbed. Consequently, the TOSCA XML format has been adapted to the specifics of the testbed. Concepts for VMs in an OpenStack private cloud and software installations with Chef were introduced. As a result, a domain-specific language based on TOSCA XML format was defined. The mappings between the TOSCA-specified entities and the Chef and OpenStack concepts were depicted in the data model that was presented previously, and the information about each deployment was retained during runtime. In addition, this task contributed to the realization of the TOSCA XML use case description by introducing certain assumptions for the usage of TOSCA specification in the context of Chef and OpenStack.

With the DSL format, all prerequisites to create packaged cloud application deployments were fulfilled. Next, a use case was defined that described a cloud-based web application consisting of four components: a HAProxy load balancer, two Apache Tomcat application servers, and a MySQL database server. The topology of the use case application was then implemented in the TOSCA-based DSL format. Also, two processes were defined within the TOSCA document (Figure 13): (1) a deployment process and (2) an un-deployment process. 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. Figure 13 depicts a graphical representation of the two BPEL processes. The TOSCA DSL description of the use case application is fully presented in the Appendix.

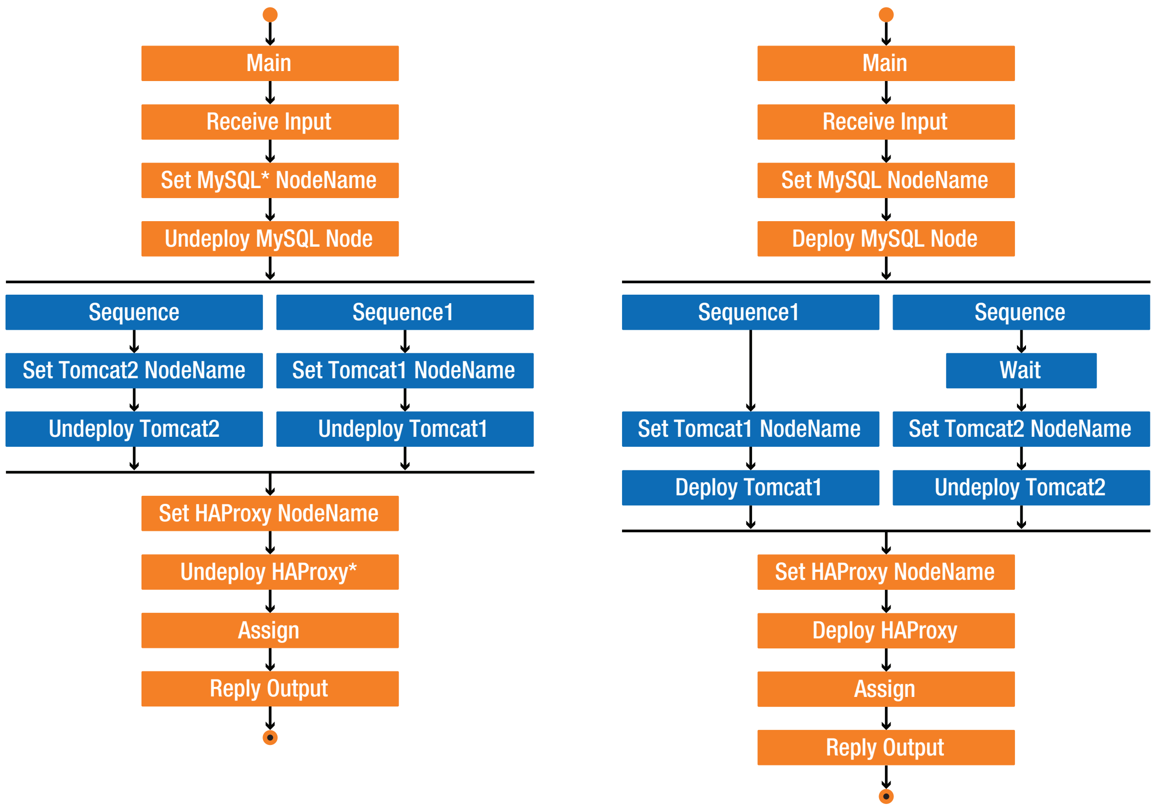
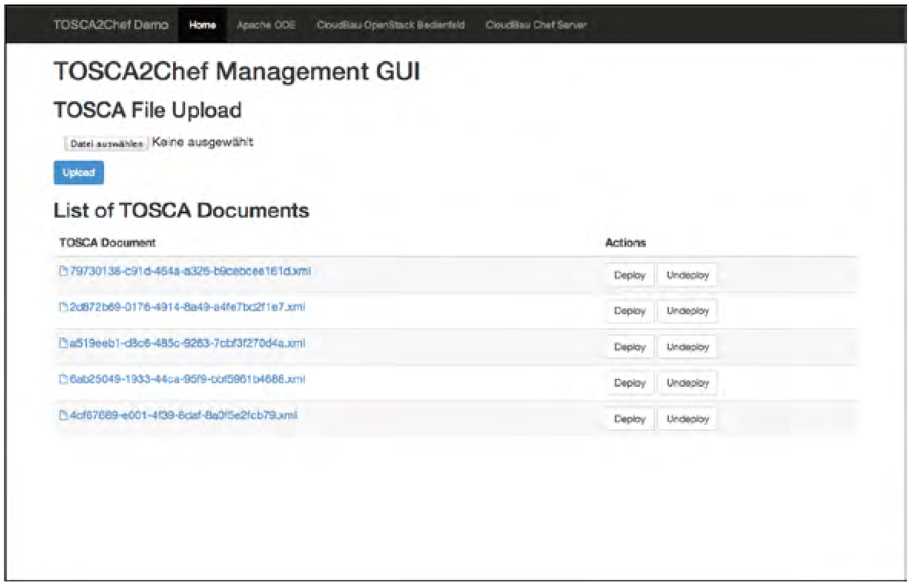


Figure 13: BPEL Deployment and Un-deployment Processes

The TOSCA2Chef runtime was developed with the knowledge of the TOSCA DSL format. However, the use case definition and the TOSCA2Chef implementation were developed in parallel over multiple iterative cycles. Ultimately, the PoC implementation, namely the TOSCA2Chef runtime and the use case TOSCA XML document, were available for an 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 14). The GUI was implemented as a web page that helps enable TOSCA XML documents to be uploaded 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 TOSCA files and provides two buttons to trigger the deployment and un-deployment processes.

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



TOSCA2Chef Management GUI

TOSCA File Upload

Keine ausgewählt

List of TOSCA Documents

TOSCA Document	Actions
79730138-c91d-464a-a326-b9cebcee161d.xml	Deploy Undeploy
2d872b69-0176-4914-8a49-a4fe7bc2f1e7.xml	Deploy Undeploy
a519eeb1-d8c6-485c-9263-7cbf3f270d4a.xml	Deploy Undeploy
6ab25049-1933-44ca-95f9-bbf5961b4686.xml	Deploy Undeploy
4cf67689-e001-4f39-8daf-8a0f5e2fcb79.xml	Deploy Undeploy

Figure 14: TOSCA2Chef Management GUI

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: www.youtube.com/watch?v=VaPADNi2IAM.

CONCLUSIONS

A capability for application packaging for cloud to enable re-usability and portability 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 concept.

Service Orchestration has to address a number of dimensions, including the following:

- Commercial Orchestration (contract and financial aspects and pre-requirements)
- Technical Service Orchestration (actual system deployments)
- Service Orchestration (configuration of systems according to SLAs with support teams' invocation, reporting, etc.)
- Consumer interface management (updating status in the Portal UI, which includes the GUI, and via the API)

Previous PoC exercises have proven that the previously mentioned dimensions may 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 DMTF's OVF and CIMI. This PoC now investigated the next layers – up into the application stack to support PaaS and SaaS.

The capability for Service Orchestration at the PaaS and SaaS levels, leveraging industry-standard frameworks and templates is still extremely limited. Each of the tooling vendors is still generally using their own frameworks and formats, and although there are historic packaging formats for the base applications themselves, they are not yet resulting in a sufficient common deployment format for Cloud Application Services, similar to: WMI (Windows* Management Instrumentation), APK (Android* Application Package), APS (Application Packaging Standard), RPM, TGZ, TAR, etc.

PoC Findings

During the development of this PoC, valuable experience was gained and significant information was gathered relating to Service Orchestration approaches using TOSCA. Those findings are summarized as follows:

- TOSCA Specification v1.0 introduces a set of new entities and concepts used in the definition of service topologies. For the cloud service specification the exact types must be defined.
- Without a realistic pool or library of resources/TOSCA-types for the nodes, relationships and artifacts, the overhead for application package development is quite large. TOSCA specification in the current form and with the current available resources is relatively abstract and therefore deriving the required cloud resources and configuration management actions becomes a challenging task, both from a development, and later from a maintenance perspective
- A domain-specific extension to TOSCA would help to define details needed to derive a system configuration. This could be achieved through the introduction of a 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 a use case is under-specified. With the current specification, multiple *NodeTypeImplementations* could be introduced, but the selection logic lies ultimately on the parser/orchestrator component.
- There are no typical models or recommended ways yet available 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 TOSCA Container implementation yet 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).

Interaction with OASIS

At the end of this PoC, a number of teleconferences were coordinated between the members of this PoC and representatives of the OASIS TOSCA working group. During those interactions, the teams shared their experiences, and the future plans and steps regarding the TOSCA specification's evolution were discussed and debated, along with other technical discussions and validation of the exercises that took place as part of the PoC.

Through this process, the team came to the conclusion that OASIS is an active group of experts from different companies and universities who spend significant effort in the ongoing shaping, evolution, and development of the TOSCA specification. They have regular weekly meetings that allow them to rapidly evolve the specification. What is more, it was commented that the specification v1.0 was the initial release that captures the basic concepts of TOSCA Service Orchestration and that in v1.1 (which is a work in progress) more clarifications and specific normative types are planned to be published. In addition, the representatives commented that the absence of TOSCA runtime API was done in order to help enable every provider to implement their own solution which could fit within the potentially available platform environments. Finally, they clearly stated that v1.1 plans to incorporate more specific resource definitions (types, relationships, etc.) and more examples and use cases that would demonstrate the applicability of TOSCA modeling in various infrastructure environments.

OpenTOSCA

As mentioned in the introduction, during the evolution of this PoC the OpenTOSCA initiative was also under development. Therefore, the results and experiences collected from the PoC did **not** include using the tools provided by OpenTOSCA, but instead we started from scratch and developed our own parsing and runtime environment. At the end of the PoC in September 2013, the initial versions of the Winery (modeling tool) and TOSCA Container were released (http://files.opentosca.de/v1). After performing an initial evaluation the team can comment that, even if the current version of the tools are a little "buggy" and subject to constant changes, they could become a useful toolset for an application developer. The GUI of the modeling tool (Figure 15) simplifies the development of a TOSCA-based service topology. The structure of the container and set up of a working TOSCA runtime is still a little complex and further support to other infrastructures might be useful (e.g., OpenStack, Chef, etc.). Overall, we believe that the OpenTOSCA initiative is an important effort and the provided tools would help the development of the PoC. On the other hand, they do not solve the shortcomings of the TOSCA v1.0 specification in terms of service topology formulation, NodeTypes definition overhead, complexity of operation description, etc.

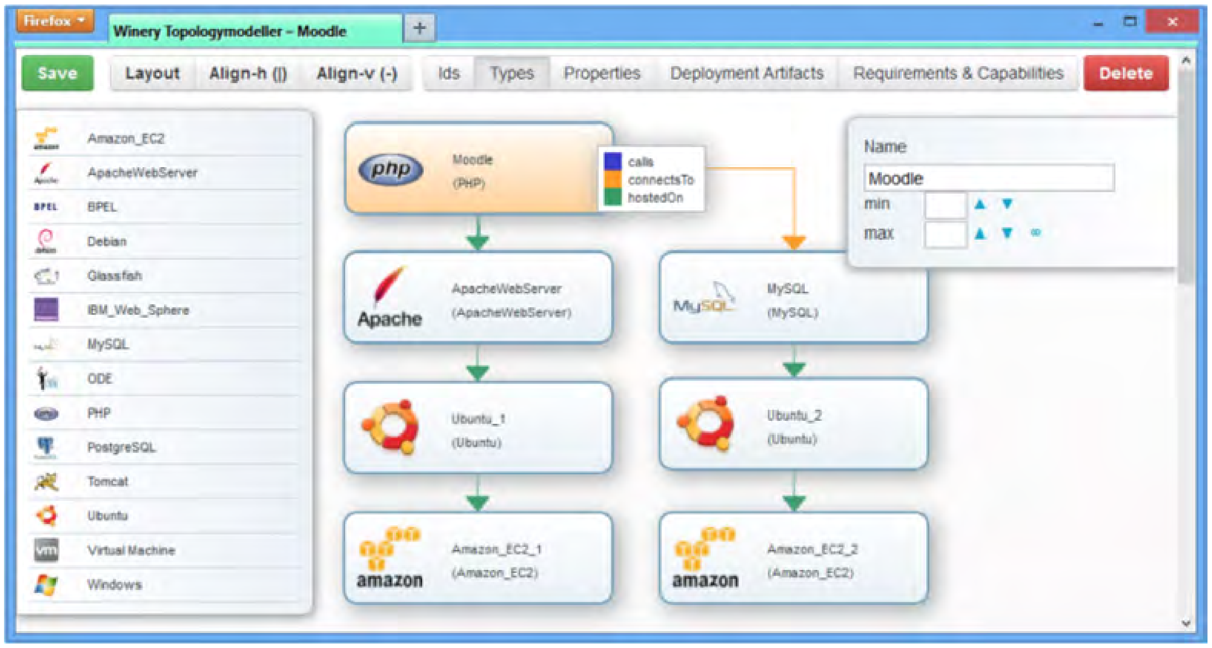


Figure 15: Winery Modeling Tool

APPENDIX

The TOSCA-based framework was created by the PoC Team who wrote this document, and can be shared hereby for reference purposes publicly.

```
<?xml version="1.0" encoding="UTF-8"?>
<Definitions
xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/tosca/ns/2011/12 http://docs.oasis-open.org/tosca/TOSCA/
v1.0/cs01/schemas/TOSCA-v1.0.xsd http://telekom.de/chefConfig Use _ Case _ ChefConfigSchema _ v2.xml http://
telekom.de/openstack Use _ Case _ OpenStackSchema _ v2.xml"
xmlns:chef="http://telekom.de/chefConfig"
xmlns:openstack="http://telekom.de/openstack"
id="Use-Case"
targetNamespace="http://www.opendatacenteralliance.org/">


<!-- Schema Definitions -->
<Types>
 <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:chef="http://telekom.de/chefConfig"
 targetNamespace="http://telekom.de/chefConfig">
 <complexType name="recipesComplexType">
 <sequence>
 <element name="recipe" maxOccurs="unbounded" minOccurs="0" type="chef:recipeComplexType" />
 </sequence>
 </complexType>

 <complexType name="recipeComplexType">
 <attribute name="name" type="string" use="required" />
 </complexType>

 <complexType name="rolesComplexType">
 <sequence>
 <element name="role" maxOccurs="unbounded" minOccurs="0" type="chef:roleComplexType" />
 </sequence>
 </complexType>

 <complexType name="roleComplexType">
 <attribute name="name" type="string" use="required" />
 </complexType>

 <complexType name="attributesComplexType">
 <sequence>
 <element name="attribute" maxOccurs="unbounded" minOccurs="0" type="chef:attributeComplexType" />
 </sequence>
 </complexType>

 <complexType name="attributeComplexType">
 <attribute name="name" type="string" use="required" />
 <attribute name="value" type="string" use="required" />
 </complexType>

 <element name="recipe" type="chef:recipeComplexType" />
 <element name="role" type="chef:roleComplexType" />
 <element name="attribute" type="chef:attributeComplexType" />
 </schema>
 <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:openstack="http://telekom.de/openstack"
 targetNamespace="http://telekom.de/openstack">
 <complexType name="imageComplexType">
 <attribute name="image" type="string" />
 </complexType>

 <complexType name="flavorComplexType">
 <attribute name="flavor" type="string" />
 </complexType>
 <element name=”image” type=”openstack:imageComplexType” />
 <element name=”flavor” type=”openstack:flavorComplexType” />
 </schema>
</Types>



<!-- NodeTypes -->

<NodeType name=”operatingsystem” abstract=”yes”>
 <InstanceStates>
 <InstanceState state=”ACTIVE” />
 <InstanceState state=”KERNEL PANIC” />
 </InstanceStates>
</NodeType>

<NodeType name=”Linux” abstract=”yes”>
 <DerivedFrom typeRef=”operatingsystem” />
 <Interfaces>
 <Interface name=”SSH”>
 <Operation name=”Connect” />
 <Operation name=”Execute” />
 <Operation name=”Put” />
 </Interface>
 </Interfaces>
</NodeType>

<NodeType name=”Ubuntu12.04”>
 <DerivedFrom typeRef=”Linux” />
</NodeType>

<NodeType name=”virtualmachineinstance” abstract=”yes”>
 <InstanceStates>
 <InstanceState state=”RUNNING” />
 <InstanceState state=”SHUTDOWN” />
 <InstanceState state=”STANDBY” />
 </InstanceStates>
 <Interfaces>
 <Interface name=”Hypervisor”>
 <Operation name=”start” />
 <Operation name=”stop” />
 <Operation name=”standby” />
 </Interface>
 <Interface name=”Chef”>
 <Operation name=”setRole”>
 <InputParameters>
  <InputParameter name=”role” type=”xs:String” />
 </InputParameters>
 </Operation>
 <Operation name=”executeRecipe”>
 <InputParameters>
  <InputParameter name=”recipe” type=”xs:String” />
 </InputParameters>
 </Operation>
 <Operation name=”setAttribute”>
 <InputParameters>
  <InputParameter name=”attributeName” type=”xs:String” />
  <InputParameter name=”attributeValue” type=”xs:String” />
 </InputParameters>
 </Operation>
 </Interface>
 </Interfaces>
</NodeType>
```

```
<NodeType name="openstackinstance" abstract="yes">
 <DerivedFrom typeRef="virtualmachineinstance" />
</NodeType>

<NodeType name="FlavorM1.small">
 <DerivedFrom typeRef="openstackinstance" />
</NodeType>

<NodeType name="Software" abstract="yes">
 <InstanceStates>
 <InstanceState state="DEPLOYED" />
 <InstanceState state="UNDEPLOYED" />
 </InstanceStates>
</NodeType>

<NodeType name="LoadBalancer" abstract="yes">
 <DerivedFrom typeRef="Software" />
 <InstanceStates>
 <InstanceState state="SERVING" />
 <InstanceState state="OFFLINE" />
 </InstanceStates>
 <Interfaces>
 <Interface name="HTTP">
 <Operation name="GET"></Operation>
 <Operation name="DELETE"></Operation>
 <Operation name="POST"></Operation>
 <Operation name="PUT"></Operation>
 </Interface>
 <Interface name="LoadBalance">
 <Operation name="Balance"></Operation>
 </Interface>
 </Interfaces>
</NodeType>

<NodeType name="HAProxy">
 <DerivedFrom typeRef="LoadBalancer" />
</NodeType>

<NodeType name="WebServer" abstract="yes">
 <DerivedFrom typeRef="Software" />
 <InstanceStates>
 <InstanceState state="SERVING" />
 <InstanceState state="OFFLINE" />
 </InstanceStates>
 <Interfaces>
 <Interface name="HTTP">
 <Operation name="GET"></Operation>
 <Operation name="DELETE"></Operation>
 <Operation name="POST"></Operation>
 <Operation name="PUT"></Operation>
 </Interface>
 </Interfaces>
</NodeType>

<NodeType name="ApacheTomcat">
 <DerivedFrom typeRef="WebServer" />
 <Interfaces>
 <Interface name="Administration">
 <Operation name="Deploy"></Operation>
 </Interface>
 </Interfaces>
</NodeType>
```

```
<NodeType name="DatabaseServer" abstract="yes">
 <DerivedFrom typeRef="Software" />
 <InstanceStates>
 <InstanceState state="SERVING" />
 <InstanceState state="OFFLINE" />
 </InstanceStates>
 <Interfaces>
 <Interface name="TCP">
 <Operation name="query">
 <InputParameters>
  <InputParameter name="SQLQueryString" type="xsd:string" />
 </InputParameters>
 </Operation>
 <Operation name="maintain" />
 </Interface>
 </Interfaces>
</NodeType>

<NodeType name="MySQL">
 <DerivedFrom typeRef="DatabaseServer" />
</NodeType>

<NodeType name="DemoWebApp">
 <InstanceStates>
 <InstanceState state="AVAILABLE" />
 <InstanceState state="UNAVAILABLE" />
 </InstanceStates>
</NodeType>

<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>

<NodeTypeImplementation name="ApacheTomcatServerImpl" nodeType="ApacheTomcat">
 <RequiredContainerFeatures>
 <RequiredContainerFeature feature="http://telekom.de/openstackcloud/" />
 <RequiredContainerFeature feature="http://telekom.de/opscodechef/" />
 </RequiredContainerFeatures>
 <DeploymentArtifacts>
 <DeploymentArtifact name="apachetomcatinstallationartifact"
 artifactType="chefinstallation" artifactRef="apachetomcatinstallationtemplate" />
 <DeploymentArtifact name="apachetomcatconfigartifact"
 artifactRef="apachetomcatconfigtemplate" artifactType="chefconfig" />
 </DeploymentArtifacts>
</NodeTypeImplementation>

<NodeTypeImplementation name="MySQLServerImpl" nodeType="MySQL">
 <RequiredContainerFeatures>
 <RequiredContainerFeature feature="http://telekom.de/openstackcloud/" />
 <RequiredContainerFeature feature="http://telekom.de/opscodechef/" />
 </RequiredContainerFeatures>
 <DeploymentArtifacts>
 <DeploymentArtifact name="mysqlinstallationartifact"
 artifactRef="mysqlinstallationtemplate" artifactType="chefinstallation" />
 </DeploymentArtifacts>
</NodeTypeImplementation>
```

```
<NodeTypeImplementation nodeType=”DemoWebApp” name=”DemoWebAppImpl”>
 <RequiredContainerFeatures>
 <RequiredContainerFeature feature=”http://telekom.de/openstackcloud/” />
 <RequiredContainerFeature feature=”http://telekom.de/opscodechef/” />
 </RequiredContainerFeatures>
 <DeploymentArtifacts>
 <DeploymentArtifact name=”demowebappdeployartifact”
 artifactType=”chefinstallation” artifactRef=”demowebappdeploytemplate” />
 </DeploymentArtifacts>
</NodeTypeImplementation>



<!-- Relationships -->

<RelationshipType name=”HostedOn”>
</RelationshipType>

<RelationshipType name=”OSHostedOnVM”>
 <DerivedFrom typeRef=”HostedOn” />
 <TargetInterfaces>
 <Interface name=”Hypervisor”>
 <Operation name=”start”></Operation>
 </Interface>
 </TargetInterfaces>
 <ValidSource typeRef=”operatingsystem” />
 <ValidTarget typeRef=”virtualmachineinstance” />
</RelationshipType>

<RelationshipType name=”SoftwareHostedOnOS”>
 <DerivedFrom typeRef=”HostedOn” />
 <TargetInterfaces>
 <Interface name=”Chef”>
 <Operation name=”executeRecipe”></Operation>
 </Interface>
 </TargetInterfaces>
 <ValidTarget typeRef=”operatingsystem” />
</RelationshipType>

<RelationshipType name=”Ubuntu12.04HostedOnM1.small”>
 <DerivedFrom typeRef=”OSHostedOnVM” />
 <ValidSource typeRef=”Ubuntu12.04” />
 <ValidTarget typeRef=”FlavorM1.small” />
</RelationshipType>

<RelationshipType name=”HAProxyHostedOnUbuntu12.04”>
 <DerivedFrom typeRef=”SoftwareHostedOnOS” />
 <ValidSource typeRef=”HAProxy” />
 <ValidTarget typeRef=”Ubuntu12.04” />
</RelationshipType>

<RelationshipType name=”ApacheTomcatHostedOnUbuntu12.04”>
 <DerivedFrom typeRef=”SoftwareHostedOnOS” />
 <ValidSource typeRef=”ApacheTomcat” />
 <ValidTarget typeRef=”Ubuntu12.04” />
</RelationshipType>

<RelationshipType name=”MySQLHostedOnUbuntu12.04”>
 <DerivedFrom typeRef=”SoftwareHostedOnOS” />
 <ValidSource typeRef=”MySQL” />
 <ValidTarget typeRef=”Ubuntu12.04” />
</RelationshipType>
```

```
<RelationshipType name="DemoWebAppHostedOnApacheTomcat">
 <DerivedFrom typeRef="HostedOn" />
 <TargetInterfaces>
 <Interface name="Administration">
 <Operation name="Deploy"></Operation>
 </Interface>
 </TargetInterfaces>
 <ValidSource typeRef="DemoWebApp" />
 <ValidTarget typeRef="ApacheTomcat" />
</RelationshipType>

<RelationshipType name="Communicate">
 <InstanceStates>
 <InstanceState state="CONNECTED" />
 <InstanceState state="DISCONNECTED" />
 </InstanceStates>
</RelationshipType>

<RelationshipType name="HAProxyCommunicateApacheTomcat">
 <DerivedFrom typeRef="Communicate" />
 <TargetInterfaces>
 <Interface name="LoadBalance">
 <Operation name="Balance"></Operation>
 </Interface>
 </TargetInterfaces>
 <ValidSource typeRef="HAProxy" />
 <ValidTarget typeRef="ApacheTomcat" />
</RelationshipType>

<RelationshipType name="DemoWebAppCommunicateMySQL">
 <DerivedFrom typeRef="Communicate" />
 <TargetInterfaces>
 <Interface name="TCP">
 <Operation name="query"></Operation>
 </Interface>
 </TargetInterfaces>
 <ValidSource typeRef="DemoWebApp" />
 <ValidTarget typeRef="MySQL" />
</RelationshipType>

<RelationshipTypeImplementation name="Ubuntu12.04HostedOnM1.smallImpl"
 relationshipType="Ubuntu12.04HostedOnM1.small">
 <ImplementationArtifacts>
 <ImplementationArtifact artifactType="virtualmachineimage"
 artifactRef="UbuntuVirtualMachineImageTemplate" interfaceName="SSH" />
 </ImplementationArtifacts>
</RelationshipTypeImplementation>

<RelationshipTypeImplementation name="HAProxyHostedOnUbuntu12.04Impl"
 relationshipType="HAProxyHostedOnUbuntu12.04" />
<RelationshipTypeImplementation name="ApacheTomcatHostedOnUbuntu12.04Impl"
 relationshipType="ApacheTomcatHostedOnUbuntu12.04" />
<RelationshipTypeImplementation name="MySQLHostedOnUbuntu12.04Impl"
 relationshipType="MySQLHostedOnUbuntu12.04"/>
<RelationshipTypeImplementation name="DemoWebAppHostedOnApacheTomcatImpl"
 relationshipType="DemoWebAppHostedOnApacheTomcat">
</RelationshipTypeImplementation>

<RelationshipTypeImplementation name="HAProxyCommunicateApacheTomcatImpl"
 relationshipType="HAProxyCommunicateApacheTomcat">
 <ImplementationArtifacts>
 <ImplementationArtifact artifactType="chefconfig" artifactRef="haproxycommunicationconfigtemplate">
</ImplementationArtifact>
 </ImplementationArtifacts>
</RelationshipTypeImplementation>
```

```
<RelationshipTypeImplementation name=”DemoWebAppCommunicateMySQLImpl”
 relationshipType=”DemoWebAppCommunicateMySQL”>
 <ImplementationArtifacts>
 <ImplementationArtifact artifactType=”chefconfig” artifactRef=”demowebappcommunicateconfigtemplate”>
</ImplementationArtifact>
 </ImplementationArtifacts>
</RelationshipTypeImplementation>



<!-- Artifacts -->

<ArtifactType name=”chefinstallation”>
 <PropertiesDefinition element=”roles” type=”chef:rolesComplexType” />
 <PropertiesDefinition element=”recipes” type=”chef:recipesComplexType” />
</ArtifactType>
<ArtifactType name=”chefconfig”>
 <PropertiesDefinition element=”attributes” type=”chef:attributesComplexType” />
</ArtifactType>

<ArtifactType name=”virtualmachineimage”>
 <PropertiesDefinition element=”image” type=”openstack:imageComplexType” />
 <PropertiesDefinition element=”flavor” type=”openstack:flavorComplexType” />
</ArtifactType>

<ArtifactTemplate id=”haproxyinstallationtemplate” type=”haproxyinstallation”>
 <Properties>
 <roles>
 <chef:role name=”haproxy” />
 </roles>
 <recipes>
 <chef:recipe name=”apt” />
 </recipes>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate id=”apachetomcatinstallationtemplate” type=”apachetomcatinstallation”>
 <Properties>
 <roles>
 <chef:role name=”tomcat-webserver” />
 </roles>
 <recipes>
 <chef:recipe name=”apt” />
 </recipes>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate id=”mysqlinstallationtemplate” type=”chefinstallation”>
 <Properties>
 <roles>
 <chef:role name=”mysql-database-server” />
 </roles>
 <recipes>
 <chef:recipe name=”apt” />
 </recipes>
 </Properties>
</ArtifactTemplate>
```

```
<ArtifactTemplate id=”haproxyconfigtemplate” type=”chefconfig”>
 <Properties>
 <attributes>
 <chef:attribute name=”incoming _ port” value=”80” />
 <chef:attribute name=”enable _ ssl” value=”false” />
 <chef:attribute name=”global _ max _ connections” value=”50000” />
 <chef:attribute name=”member _ max _ connections” value=”50000” />
 </attributes>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate id=”apachetomcatconfigtemplate” type=”chefconfig”>
 <Properties>
 <attributes>
 <chef:attribute name=”port” value=”8080” />
 <chef:attribute name=”java _ options” value=”-Xmx768M” />
 <chef:attribute name=”loglevel” value=”INFO” />
 </attributes>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate id=”mysqlconfigtemplate” type=”chefconfig”>
 <Properties>
 <attributes>
 <chef:attribute name=”username” value=”mysql” />
 <chef:attribute name=”password” value=”1234” />
 </attributes>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate id=”UbuntuVirtualMachineImageTemplate” type=”virtualmachine”>
 <Properties>
 <openstack>
 <openstack:image image=”8b014744-a0b7-46b4-8ecd-d9d666eec570” />
 <openstack:flavor flavor=”7” />
 </openstack>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate type=”chefinstallation” id=”demowebappdeploytemplate”>
 <Properties>
 <roles>
 <chef:role name=”demo-webapp” />
 </roles>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate type=”chefconfig” id=”demowebappconfigtemplate”>
 <Properties>
 <attributes>
 <chef:attribute name=”db _ name” value=”tcom” />
 <chef:attribute name=”db _ user” value=”mysql” />
 <chef:attribute name=”db _ password” value=”1234” />
 </attributes>
 </Properties>
</ArtifactTemplate>

<ArtifactTemplate type=”chefconfig” id=”haproxycommunicationconfigtemplate”>
 <Properties>
 <attributes>
 <chef:attribute name=”server _ pool” value=”webserver” />
 </attributes>
 </Properties>
</ArtifactTemplate>
```

```
<ArtifactTemplate type="chefconfig"
 id="demowebappcommunicationconfigtemplate">
 <Properties>
 <recipes>
 <chef:recipes name="demo-webapp-config" />
 </recipes>
 </Properties>
</ArtifactTemplate>



<!-- Service Template -->

<ServiceTemplate id="TelecomUseCaseServiceTemplate">
 <Tags>
 <Tag name="authors" value="Michael Menzel, Gregory Katsaros" />
 <Tag name="organization" value="FZI Forschungszentrum Informatik at Karlsruhe Institute of Technology (KIT)" />
 </Tags>

 <TopologyTemplate>
 <NodeTemplate type="FlavorM1.small" id="VMNode4HAProxy" />
 <NodeTemplate type="FlavorM1.small" id="VMNode4Tomcat1" />
 <NodeTemplate type="FlavorM1.small" id="VMNode4Tomcat2" />
 <NodeTemplate type="FlavorM1.small" id="VMNode4MySQL" />

 <NodeTemplate type="Ubuntu12.04" id="UbuntuNode4HAProxy" />
 <NodeTemplate type="Ubuntu12.04" id="UbuntuNode4Tomcat1" />
 <NodeTemplate type="Ubuntu12.04" id="UbuntuNode4Tomcat2" />
 <NodeTemplate type="Ubuntu12.04" id="UbuntuNode4MySQL" />

 <NodeTemplate type="HAProxy" id="HAProxyNode" />

 <NodeTemplate type="ApacheTomcat" id="Tomcat1Node" />
 <NodeTemplate type="ApacheTomcat" id="Tomcat2Node" />

 <NodeTemplate type="MySQL" id="MySQLNode" />

 <NodeTemplate type="DemoWebApp" id="DemoWebApp1Node" />
 <NodeTemplate type="DemoWebApp" id="DemoWebApp2Node" />

 <RelationshipTemplate type="Ubuntu12.04HostedOnM1.small" id="UbuntuNode4HAProxyHostedOnVMNode4HAProxy">
 <SourceElement ref="UbuntuNode4HAProxy" />
 <TargetElement ref="VMNode4HAProxy" />
 </RelationshipTemplate>

 <RelationshipTemplate type="Ubuntu12.04HostedOnM1.small" id="UbuntuNode4Tomcat1HostedOnVMNode4Tomcat1">
 <SourceElement ref="UbuntuNode4Tomcat1" />
 <TargetElement ref="VMNode4Tomcat1" />
 </RelationshipTemplate>

 <RelationshipTemplate type="Ubuntu12.04HostedOnM1.small" id="UbuntuNode4Tomcat2HostedOnVMNode4Tomcat2">
 <SourceElement ref="UbuntuNode4Tomcat2" />
 <TargetElement ref="VMNode4Tomcat2" />
 </RelationshipTemplate>

 <RelationshipTemplate type="Ubuntu12.04HostedOnM1.small" id="UbuntuNode4MySQLHostedOnVMNode4MySQL">
 <SourceElement ref="UbuntuNode4MySQL" />
 <TargetElement ref="VMNode4MySQL" />
 </RelationshipTemplate>

 <RelationshipTemplate type="HAProxyHostedOnUbuntu12.04" id="HAProxyNodeHostedOnUbuntuNode4HAProxy">
 <SourceElement ref="HAProxyNode" />
 <TargetElement ref="UbuntuNode4HAProxy" />
 </RelationshipTemplate>
 <RelationshipTemplate type=”ApacheTomcatHostedOnUbuntu12.04” id=”Tomcat1NodeHostedOnUbuntuNode4Tomcat1”>
 <SourceElement ref=”Tomcat1Node” />
 <TargetElement ref=”UbuntuNode4Tomcat1” />
 </RelationshipTemplate>

 <RelationshipTemplate type=”ApacheTomcatHostedOnUbuntu12.04” id=”Tomcat2NodeHostedOnUbuntuNode4Tomcat2”>
 <SourceElement ref=”Tomcat2Node” />
 <TargetElement ref=”UbuntuNode4Tomcat2” />
 </RelationshipTemplate>

 <RelationshipTemplate type=”MySQLHostedOnUbuntu12.04” id=”MySQLNodeHostedOnUbuntuNode4MySQL”>
 <SourceElement ref=”MySQLNode” />
 <TargetElement ref=”UbuntuNode4MySQL” />
 </RelationshipTemplate>

 <RelationshipTemplate type=”DemoWebAppHostedOnApacheTomcat” id=”DemoWebApp1HostedOnTomcat1Node”>
 <SourceElement ref=”DemoWebApp1Node” />
 <TargetElement ref=”Tomcat1Node” />
 </RelationshipTemplate>

 <RelationshipTemplate type=”DemoWebAppHostedOnApacheTomcat” id=”DemoWebApp2HostedOnTomcat2Node”>
 <SourceElement ref=”DemoWebApp2Node” />
 <TargetElement ref=”Tomcat2Node” />
 </RelationshipTemplate>

 <RelationshipTemplate id=”HAProxyNodeCommunicateTomcat1Node” type=”HAProxyCommunicateApacheTomcat”>
 <SourceElement ref=”HAProxyNode” />
 <TargetElement ref=”Tomcat1Node” />
 </RelationshipTemplate>

 <RelationshipTemplate id=”HAProxyNodeCommunicateTomcat2Node” type=”HAProxyCommunicateApacheTomcat”>
 <SourceElement ref=”HAProxyNode” />
 <TargetElement ref=”Tomcat2Node” />
 </RelationshipTemplate>

 <RelationshipTemplate id=”DemoWebApp1NodeCommunicateMySQLNode” type=”DemoWebAppCommunicateMySQL”>
 <SourceElement ref=”DemoWebApp1Node” />
 <TargetElement ref=”MySQLNode” />
 </RelationshipTemplate>

 <RelationshipTemplate id=”DemoWebApp2NodeCommunicateMySQLNode” type=”DemoWebAppCommunicateMySQL”>
 <SourceElement ref=”DemoWebApp2Node” />
 <TargetElement ref=”MySQLNode” />
 </RelationshipTemplate>

 </TopologyTemplate>


<Plans>

 <Plan id=”DeploymentPlan” planType=”http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan”
 planLanguage=”http://docs.oasis-open.org/wsbpel/2.0/process/executable”>

 <InputParameters>
 <parameter name=”input” type=”xsd:String”/>
 </InputParameters>

 <OutputParameters>
 </OutputParameters>
```

```
<PlanModel>
<bpel:process
   name="TOSCA2ChefDeploymentProcess"
   targetNamespace="http://t-systems.com/TOSCA2ChefDeploymentProcess"
   suppressJoinFailure="yes"
   xmlns:tns="http://t-systems.com/TOSCA2ChefDeploymentProcess"
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:ns1="http://www.w3.org/2001/XMLSchema"
   xmlns:ns="http://webservice.container.tosca.laboratories.telekom.com/">

 <!-- Import the client WSDL -->

 <bpel:import namespace="http://webservice.container.tosca.laboratories.telekom.com/"
  location="TOSCA2ChefWS.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import>

 <bpel:import location="TOSCA2ChefDeploymentProcessArtifacts.wsdl"
  namespace="http://t-systems.com/TOSCA2ChefDeploymentProcess"
  importType="http://schemas.xmlsoap.org/wsdl/" />



 <!-- ================================================================= -->
 <!-- PARTNERLINKS                                                      -->
 <!-- List of services participating in this BPEL process               -->
 <!-- ================================================================= -->

 <bpel:partnerLinks>
  <!-- The 'client' role represents the requester of this service. -->
  <bpel:partnerLink name="client"
      partnerLinkType="tns:TOSCA2ChefDeploymentProcess"
      myRole="TOSCA2ChefDeploymentProcessProvider" />

  <bpel:partnerLink name="TOSCA2ChefPL" partnerLinkType="tns:TOSCA2ChefWSPLT"
   partnerRole="TOSCA2ChefWSPLTRole"></bpel:partnerLink>
 </bpel:partnerLinks>



 <!-- ================================================================= -->
 <!-- VARIABLES                                                         -->
 <!-- List of messages and XML documents used within this BPEL process  -->
 <!-- ================================================================= -->

 <bpel:variables>
  <!-- Reference to the message passed as input during initiation         -->
  <bpel:variable name="input" messageType="tns:TOSCA2ChefDeploymentProcessRequestMessage">
  </bpel:variable>

  <!-- Reference to the message that will be returned to the requester    -->
  <bpel:variable name="output" messageType="tns:TOSCA2ChefDeploymentProcessResponseMessage">
  </bpel:variable>

  <bpel:variable name="TOSCA2ChefWSPLResponse" messageType="ns:deployNodeByNameResponse">
  </bpel:variable>

  <bpel:variable name="TOSCA2ChefWSPLRequest" messageType="ns:deployNodeByName">
  </bpel:variable>

  <bpel:variable name="DeployTomcat1Input" messageType="ns:deployNodeByName"></bpel:variable>
  <bpel:variable name="DeployTomcat2Input" messageType="ns:deployNodeByName"></bpel:variable>
  </bpel:variables>
```

```
<!-- ================================================================== -->
<!-- ORCHESTRATION LOGIC                                                -->
<!-- Set of activities coordinating the flow of messages across the     -->
<!-- services integrated within this business process                  -->
<!-- ================================================================== -->

<bpel:sequence name="main">

<!-- Receive input from requester. Note: This maps to operation defined in HelloBPELProcess.wsdl -->

 <bpel:receive name="receiveInput" partnerLink="client" portType="tns:TOSCA2ChefDeploymentProcess"
  operation="process" variable="input" createInstance="yes" />

<!-- Generate reply to synchronous request -->

 <bpel:assign validate="no" name="SetMySQLNodename">
  <bpel:copy>
   <bpel:from>
    <bpel:literal xml:space="preserve">MySQLNode</bpel:literal>
   </bpel:from>
   <bpel:to part="arg0" variable="TOSCA2ChefWSPLRequest"></bpel:to>
 </bpel:copy>

  <bpel:copy>
   <bpel:from>
    <bpel:literal xml:space="preserve">2d872b69-0176-4914-8a49-a4fe7bd2f1e7</bpel:literal>
   </bpel:from>
   <bpel:to part="arg1" variable="TOSCA2ChefWSPLRequest"></bpel:to>
  </bpel:copy>
 </bpel:assign>
 <bpel:invoke name="DeployMySQLNode" partnerLink="TOSCA2ChefPL" operation="deployNodeByName"
  portType="ns:TOSCA2ChefWebserviceBlockingWithID" inputVariable="TOSCA2ChefWSPLRequest"
  outputVariable="TOSCA2ChefWSPLResponse"></bpel:invoke>
 <bpel:flow name="Flow"><bpel:sequence name="Sequence">
   <bpel:wait name="Wait">
    <bpel:for>'P0Y0M0DT0H0M10S'</bpel:for>
   </bpel:wait>
   <bpel:assign validate="no" name="SetTomcat2Nodename">
    <bpel:copy>
     <bpel:from>
      <bpel:literal xml:space="preserve">DemoWebApp2Node</bpel:literal>
     </bpel:from>
     <bpel:to part="arg0" variable="DeployTomcat2Input"></bpel:to>
    </bpel:copy>

    <bpel:copy>
     <bpel:from>
      <bpel:literal xml:space="preserve">2d872b69-0176-4914-8a49-a4fe7bd2f1e7</bpel:literal>
     </bpel:from>
     <bpel:to part="arg1" variable="DeployTomcat2Input"></bpel:to>
    </bpel:copy>

   </bpel:assign>
   <bpel:invoke name="DeployTomcat2" partnerLink="TOSCA2ChefPL" operation="deployNodeByName"
    portType="ns:TOSCA2ChefWebserviceBlockingWithID" inputVariable="DeployTomcat2Input"
    outputVariable="TOSCA2ChefWSPLResponse"></bpel:invoke>
  </bpel:sequence><bpel:sequence name="Sequence1">
   <bpel:assign validate="no" name="SetTomcat1Nodename">
    <bpel:copy>
     <bpel:from>
      <bpel:literal xml:space="preserve">DemoWebApp1Node</bpel:literal>
     </bpel:from>
     <bpel:to part="arg0" variable="DeployTomcat1Input"></bpel:to>
    </bpel:copy>
```

```
      <bpel:copy>
       <bpel:from>
        <bpel:literal xml:space=”preserve”>2d872b69-0176-4914-8a49-a4fe7bd2f1e7</bpel:literal>
       </bpel:from>
       <bpel:to part=”arg1” variable=”DeployTomcat1Input”></bpel:to>
      </bpel:copy>

     </bpel:assign>
     <bpel:invoke name=”DeployTomcat1” partnerLink=”TOSCA2ChefPL” operation=”deployNodeByName”
      portType=”ns:TOSCA2ChefWebserviceBlockingWithID” inputVariable=”DeployTomcat1Input”
      outputVariable=”TOSCA2ChefWSPLResponse”></bpel:invoke>
    </bpel:sequence></bpel:flow>

   <bpel:assign validate=”no” name=”SetHAProxyNodename”><bpel:copy>
     <bpel:from>
      <bpel:literal xml:space=”preserve”>HAProxyNode</bpel:literal>
     </bpel:from>
     <bpel:to part=”arg0” variable=”TOSCA2ChefWSPLRequest”></bpel:to>
    </bpel:copy>

    <bpel:copy>
     <bpel:from>
      <bpel:literal xml:space=”preserve”>2d872b69-0176-4914-8a49-a4fe7bd2f1e7</bpel:literal>
     </bpel:from>
     <bpel:to part=”arg1” variable=”TOSCA2ChefWSPLRequest”></bpel:to>
    </bpel:copy>
   </bpel:assign>
   <bpel:invoke name=”DeployHAProxy” partnerLink=”TOSCA2ChefPL” operation=”deployNodeByName”
    portType=”ns:TOSCA2ChefWebserviceBlockingWithID” inputVariable=”TOSCA2ChefWSPLRequest”
    outputVariable=”TOSCA2ChefWSPLResponse”></bpel:invoke>

   <bpel:assign validate=”no” name=”Assign”>

    <bpel:copy>
     <bpel:from><bpel:literal><tns:TOSCA2ChefDeploymentProcessResponse
      xmlns:tns=”http://t-systems.com/TOSCA2ChefDeploymentProcess”
      xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>
  <tns:result>tns:result</tns:result>
 </tns:TOSCA2ChefDeploymentProcessResponse>
 </bpel:literal></bpel:from>
     <bpel:to variable=”output” part=”payload”></bpel:to>
    </bpel:copy>
    <bpel:copy>
     <bpel:from part=”payload” variable=”input”>
      <bpel:query queryLanguage=”urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0”>
      <![CDATA[tns:input]]></bpel:query>
     </bpel:from>
     <bpel:to part=”payload” variable=”output”>
      <bpel:query queryLanguage=”urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0”>
      <![CDATA[tns:result]]></bpel:query>
     </bpel:to>
    </bpel:copy>
   </bpel:assign>
   <bpel:reply name=”replyOutput” partnerLink=”client” portType=”tns:TOSCA2ChefDeploymentProcess”
    operation=”process” variable=”output” />

  </bpel:sequence>
 </bpel:process>

 </PlanModel>
 </Plan>
```

```
<Plan id="UndeploymentPlan"
 planType="http://docs.oasis-open.org/tosca/ns/2011/12/PlanTypes/BuildPlan"
 planLanguage="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
<InputParameters>
<parameter name="input" type="xsd:String"/>
</InputParameters>
<OutputParameters>
</OutputParameters>

<PlanModel>
<bpel:process name="TOSCA2ChefUndeploymentProcess"
   targetNamespace="http://t-systems.com/TOSCA2ChefUndeploymentProcess"
   suppressJoinFailure="yes"
   xmlns:tns="http://t-systems.com/TOSCA2ChefUndeploymentProcess"
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:ns1="http://www.w3.org/2001/XMLSchema"
   xmlns:ns="http://webservice.container.tosca.laboratories.telekom.com/">

 <!-- Import the client WSDL -->

 <bpel:import namespace="http://webservice.container.tosca.laboratories.telekom.com/"
  location="TOSCA2ChefWS.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import>
 <bpel:import location="TOSCA2ChefUndeploymentProcessArtifacts.wsdl"
  namespace="http://t-systems.com/TOSCA2ChefUndeploymentProcess"
  importType="http://schemas.xmlsoap.org/wsdl/" />



 <!-- ================================================================= -->
 <!-- PARTNERLINKS                                                      -->
 <!-- List of services participating in this BPEL process               -->
 <!-- ================================================================= -->

 <bpel:partnerLinks>
  <!-- The 'client' role represents the requester of this service. -->
  <bpel:partnerLink name="client" partnerLinkType="tns:TOSCA2ChefUndeploymentPLT"
   myRole="TOSCA2ChefPLTRole" />

  <bpel:partnerLink name="TOSCA2ChefPL" partnerLinkType="tns:TOSCA2ChefWSPLT"
   partnerRole="TOSCA2ChefWSPLTRole"></bpel:partnerLink>
 </bpel:partnerLinks>



 <!-- ================================================================= -->
 <!-- VARIABLES                                                         -->
 <!-- List of messages and XML documents used within this BPEL process  -->
 <!-- ================================================================= -->

 <bpel:variables>
  <!-- Reference to the message passed as input during initiation -->
  <bpel:variable name="input" messageType="tns:TOSCA2ChefUndeploymentProcessRequestMessage">
  </bpel:variable>

  <!--  Reference to the message that will be returned to the requester  -->
  <bpel:variable name="output" messageType="tns:TOSCA2ChefUndeploymentProcessResponseMessage">
  </bpel:variable>

  <bpel:variable name="UndeployTomcat1Input" messageType="ns:deleteNode"></bpel:variable>
  <bpel:variable name="UndeployTomcat2Input" messageType="ns:deleteNode"></bpel:variable>
  <bpel:variable name="UndeployHAProxyInput" messageType="ns:deleteNode"></bpel:variable>
  <bpel:variable name="UndeployMySQLInput" messageType="ns:deleteNode"></bpel:variable>
  <bpel:variable name="TOSCA2ChefPLResponse" messageType="ns:deleteNodeResponse"></bpel:variable>
  <bpel:variable name="TOSCA2ChefPLRequest" messageType="ns:deleteNode"></bpel:variable>
 </bpel:variables>
```

```
<!-- ================================================================= -->
<!-- ORCHESTRATION LOGIC                                                -->
<!-- Set of activities coordinating the flow of messages across the    -->
<!-- services integrated within this business process                  -->
<!-- ================================================================= -->

<bpel:sequence name="main">

 <!-- Receive input from requester. Note: This maps to operation defined in HelloBPELProcess.wsdl -->
 <bpel:receive name="receiveInput" partnerLink="client"
  portType="tns:TOSCA2ChefUndeploymentProcess" operation="process" variable="input"
  createInstance="yes" />

 <!-- Generate reply to synchronous request -->

 <bpel:assign validate="no" name="SetMySQLNodename"><bpel:copy><bpel:from>
    <bpel:literal xml:space="preserve">MySQLNode</bpel:literal>
   </bpel:from>
   <bpel:to part="arg0" variable="UndeployHAProxyInput"></bpel:to>
  </bpel:copy>
 </bpel:assign>

 <bpel:invoke name="UndeployMySQLNode" partnerLink="TOSCA2ChefPL" operation="deleteNode"
  portType="ns:TOSCA2ChefWebserviceBlockingWithID" inputVariable="UndeployHAProxyInput"
  outputVariable="TOSCA2ChefPLResponse">
 </bpel:invoke>

 <bpel:flow name="Flow"><bpel:sequence name="Sequence">
   <bpel:assign validate="no" name="SetTomcat2Nodename">
    <bpel:copy>
     <bpel:from>
      <bpel:literal xml:space="preserve">DemoWebApp2Node</bpel:literal>
     </bpel:from>
     <bpel:to part="arg0" variable="UndeployTomcat2Input"></bpel:to>
    </bpel:copy>
   </bpel:assign>
   <bpel:invoke name="UndeployTomcat2" partnerLink="TOSCA2ChefPL" operation="deleteNode"
    portType="ns:TOSCA2ChefWebserviceBlockingWithID" inputVariable="UndeployTomcat2Input"
    outputVariable="TOSCA2ChefPLResponse"></bpel:invoke>
  </bpel:sequence><bpel:sequence name="Sequence1">
   <bpel:assign validate="no" name="SetTomcat1Nodename">
    <bpel:copy>
     <bpel:from>
      <bpel:literal xml:space="preserve">DemoWebApp1Node</bpel:literal>
     </bpel:from>
     <bpel:to part="arg0" variable="UndeployTomcat1Input"></bpel:to>
    </bpel:copy>
   </bpel:assign>

   <bpel:invoke name="UndeployTomcat1" partnerLink="TOSCA2ChefPL" operation="deleteNode"
    portType="ns:TOSCA2ChefWebserviceBlockingWithID" inputVariable="UndeployTomcat1Input"
    outputVariable="TOSCA2ChefPLResponse"></bpel:invoke>
  </bpel:sequence></bpel:flow>

 <bpel:assign validate="no" name="SetHAProxyNodename"><bpel:copy>
   <bpel:from>
    <bpel:literal xml:space="preserve">HAProxyNode</bpel:literal>
   </bpel:from>
   <bpel:to part="arg0" variable="UndeployHAProxyInput"></bpel:to>
  </bpel:copy>

 </bpel:assign>
 <bpel:invoke name="UndeployHAProxy" partnerLink="TOSCA2ChefPL" operation="deleteNode"
  portType="ns:TOSCA2ChefWebserviceBlockingWithID" inputVariable="UndeployHAProxyInput"
  outputVariable="TOSCA2ChefPLResponse"></bpel:invoke>
```

```
   <bpel:assign validate=”no” name=”Assign”>

    <bpel:copy>
     <bpel:from><bpel:literal><tns2:TOSCA2ChefUndeploymentProcessResponse
      xmlns:tns2=”http://t-systems.com/TOSCA2ChefUndeploymentProcess”
      xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>
  <tns2:result>tns2:result</tns2:result>
 </tns2:TOSCA2ChefUndeploymentProcessResponse>

 </bpel:literal></bpel:from>
     <bpel:to variable=”output” part=”payload”></bpel:to>
    </bpel:copy>
    <bpel:copy>
     <bpel:from part=”payload” variable=”input”>
      <bpel:query queryLanguage=”urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0”>
      <![CDATA[tns:input]]></bpel:query>
     </bpel:from>
     <bpel:to part=”payload” variable=”output”>
      <bpel:query queryLanguage=”urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0”>
      <![CDATA[tns:result]]></bpel:query>
     </bpel:to>
    </bpel:copy>
   </bpel:assign>
   <bpel:reply name=”replyOutput” partnerLink=”client” portType=”tns:TOSCA2ChefUndeploymentProcess”
    operation=”process” variable=”output” />

  </bpel:sequence>
 </bpel:process>

 </PlanModel>
 </Plan>
 </Plans>
</ServiceTemplate>

</Definitions>
```