Institute of Architecture of Application Systems

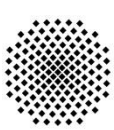
OpenTOSCA Injector: Vertical and Horizontal Topology Model Injection

Karoline Saatkamp, Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Michael Zimmermann

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
[firstname.lastname]@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Saatkamp2017_OpenTOSCAInjector,
  author    = {Karoline Saatkamp and Uwe Breitenb{\"u}cher and K{\'a}lm{\'a}n
               K{\'e}pes and Frank Leymann and Michael Zimmermann},
  title     = {OpenTOSCA Injector: Vertical and Horizontal Topology Model
               Injection},
  booktitle = {Service-Oriented Computing – ICSOC 2017 Workshops},
  year      = {2018},
  pages     = {379--383},
  doi       = {10.1007/978-3-319-91764-1},
  series    = {Lecture Notes in Computer Science (LNCS)},
  volume    = {10797},
  publisher = {Springer International Publishing}
}
```

University of Stuttgart
Germany

OpenTOSCA Injector: Vertical and Horizontal Topology Model Injection

Karoline Saatkamp, Uwe Breitenbücher, Kálmán Képes,
Frank Leymann, and Michael Zimmermann

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

Abstract. The automation of application deployments is supported by various technologies. The TOSCA standard facilitates to describe application deployments in a portable manner by modeling application structures as topology models. The final structure often depends on the target environment and is, therefore, not always known at modeling time. However, a manual adaptation is error-prone and time-consuming. In this paper, we demonstrate the OpenTOSCA Injector for an automated completion of topology models: the extended TOSCA runtime OpenTOSCA for an automated injection and deployment is presented.

Keywords: TOSCA, Deployment Model, Completion Automation

1 Introduction and Motivation

In recent years, several technologies and standards were developed to automate the deployment of cloud applications. This includes configuration management technologies such as Chef, container technologies such as Docker, and standards such as the Topology and Orchestration Specification for Cloud Applications (TOSCA) [6]. TOSCA is an OASIS standard that enables to define application deployments by topology models and management plans, which can be executed automatically by a TOSCA runtime, e.g., the OpenTOSCA container [1].

A topology model describes the application components and their relations. This includes application-specific components, such as PHP applications or databases, middleware, and infrastructure components, such as web servers or virtual machines. Thereby, application deployments can be described in a vendor-independent and portable manner. However, the available middleware, infrastructure, as well as application-specific components can differ between environments. When, for example, application deployments are provided for third parties or parts of the IT infrastructure are outsourced, the target environment is not known in advance. Thus, the final topology model is not known at modeling time. However, the manual adaptation for each target environment is time-consuming and error-prone [4]. To enable an environment-independent modeling via incomplete topology models and an automated environment-specific injection

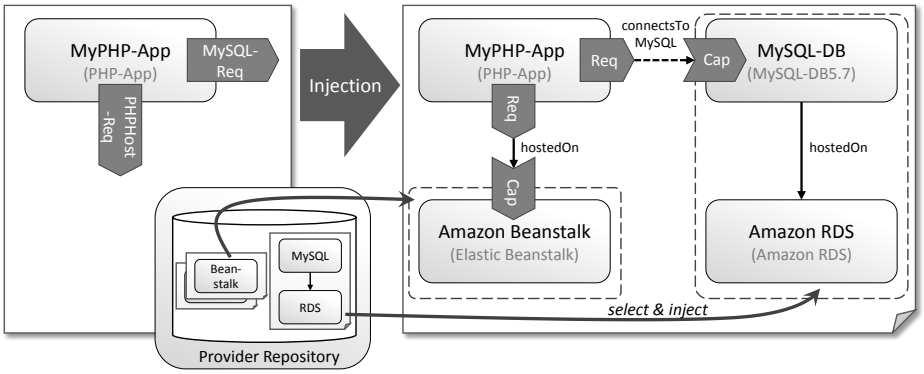


Fig. 1. Topology model with open requirements (left) and injected components (right)

of components during deployment time, the *OpenTOSCA Injector* is developed: infrastructure components (vertical injection) as well as, e.g., data storage stacks (horizontal injection) are selected and injected to complete formerly incomplete topology models.

2 TOSCA Fundamentals and Injection Concept

As already mentioned TOSCA is an OASIS standard that enables to describe the automated deployment of applications in a vendor-independent and portable manner. Several TOSCA runtimes to process TOSCA models are already developed such as Cloudify[1], Apache ARIA TOSCA[2], and the OpenTOSCA container[3]. In the following all TOSCA concepts relevant for the OpenTOSCA Injector are introduced. More details about TOSCA can be found in the specification [6].

The structure of an application can be described as *Topology Template*, which is a directed and weighted multigraph as depicted in Fig. 1 on the right. The components are modeled as *Node Templates*, e.g., MyPHP-App, and the relations between them as *Relationship Templates* such as hostedOn. Their semantic is defined by *Node Types*, e.g., PHP-App, and *Relationship Types*, respectively. Types can be derived from other types, thus, inheritance hierarchies can be defined. For Relationship Types valid target and source elements are specified, which can be Node Types or *Requirement Types* and *Capability Types*. For each Requirement Type, exactly one *requiredCapabilityType* is defined, i.e., each Capability of this type can be matched to a Requirement of the respective Requirement Type. *Requirements* and *Capabilities* of these types can be attached to Node Templates. Thus, the matching between Requirements and Capabilities and hence between Node Templates is realized, which is the basis for the OpenTOSCA Injector.

[1] http://cloudify.co/
[2] http://ariatosca.incubator.apache.org/
[3] http://www.opentosca.org/

The left side of Fig. 1 shows an incomplete topology with two open Requirements. The Requirements *PHPHost-Req* and *MySQL-Req* require Node Templates with matching Capabilities. Possible suitable Node Templates are stored in a local *Provider Repository* in which the respective owner can add all available components in the environment, such as specific infrastructure components. However, in future work also the linkage to public repositories should be enabled. With the OpenTOSCA Injector, not only single Node Templates but also topology fragments can be injected [4]. As shown in Fig. 1 on the right, a topology fragment could consist of a *MySQL-DB* and an *Amazon RDS* component, which is injected based on the matching between the Requirement *MySQL-Req* and the specified requiredCapabilityType, e.g., *MySQL-Cap* attached to the *MySQL-DB*.

For each match a suitable Relationship Type has to be found to connect the matched Node Templates. This, for example, could be a *hostedOn* or *connectsTo* relation. The suitable Relationship Type is determined by the assigned Requirement and Capability. However, specific types such as *connectsToMySQL* are not always available in the target environment. For this, TOSCA base types are used: the *hostedOn* and the *connectsTo* Relationship Type [7]. In any case, one of these base types is selected, because of the predefined inheritance hierarchy of Capability Types. After the Node Templates or topology fragments are injected with suitable Relationship Types, the topology is complete and deployable. We implemented the described injection concept and demonstrate it with the OpenTOSCA Injector, which extends the existing OpenTOSCA container.

3 System Architecture and Demonstration

The OpenTOSCA container is a TOSCA runtime supporting the imperative and declarative processing of TOSCA models for an automated deployment [2]. For the imperative processing the management plans are explicitly defined, whereas at the declarative processing the deployment logic is inferred from the topology model. Our demonstration is based on declarative provisioning modeling and

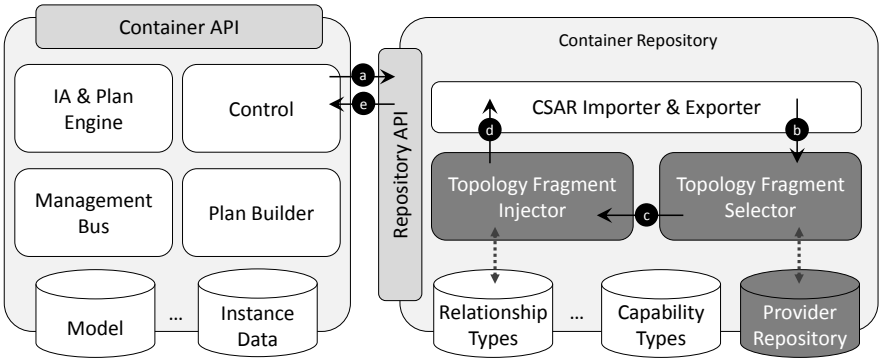


Fig. 2. Extended OpenTOSCA System Architecture and Processing Overview

management plans are not explicitly considered. Besides the actual purpose of the OpenTOSCA container to deploy cloud application, it is also used for the automated deployment of use cases of the 4th Industrial Revolution [3] and IoT scenarios with different messaging middleware systems [9,10]. In this demonstration the OpenTOSCA container is used to automatically complete and deploy topology models for different deployment environments.

In Fig. 2 the extended OpenTOSCA system architecture[4] is depicted. While the left hand side shows the existing OpenTOSCA components required for the deployment, the right hand side shows the *Container Repository* extending the existing runtime. The Container API is used to upload topology models and all related artifacts, such as JAR files or scripts for the deployment. The *Control* component is responsible for interpreting the topology and tracking the process. For a declarative processing, the *Plan Builder* generates plans based on the topology model. The operations invoked by the plans need Implementation Artifacts (IA) to install and start the application's components. They are part of the upload and processed by the *IA Engine*, while the plans are processed by the *Plan Engine*. With the *Management Bus*, plans finally invoke different kinds of management operations for the deployment. All data required during the deployment, e.g., model information, and after the deployment such as the instance data, are stored in databases.

For the demonstration of the injection the *Container Repository* is essential. Its source code is based on the *Eclipse Winery*[5], a modeling tool for TOSCA [5]. Because the injection affects the topology model, the existing Winery capabilities to deal with topology model elements is utilized. The *Topology Fragment Injector*, the *Topology Fragment Selector* component, and the *Provider Repository* extend the existing Winery source code to use it as Container Repository for the injection. For the injection, the incomplete topology as depicted in Fig. 1 on the left is uploaded to the Container API and forwarded to the Control component. It checks the topology model for open requirements and in case open requirements are contained, an injection request is sent to the Container Repository (cf. (a) in Fig. 2). The Topology Fragment Selector browses the Provider Repository for topology fragments with matching Capabilities (cf. (b) in Fig. 2). For multiple injection options, the user can select the preferred fragment. After the selection, suitable Relationship Templates are determined based on the Requirements and Capabilities, and used to inject the topology fragments in the model (cf. (c) in Fig. 2). The completed topology model as presented in Fig. 1 is exported and the Control component starts the deployment of the application (cf. (d) in Fig. 2).

The demonstrated OpenTOSCA Injector implements the TOSCA concept for Requirement and Capability matching in an automated manner. It facilitates, beyond the general matching of Capabilities, the injection of whole topology fragments. The objective is to model an incomplete topology model with defined requirements which is completed depending on the specific deployment environment, e.g., a factory, company, or public cloud provider. The Injector can be used

[4] https://github.com/OpenTOSCA
[5] https://github.com/eclipse/winery

for the completion by, e.g., different infrastructure components (vertical injection) such as an OpenStack or vSphere depending on the available infrastructure as well as for the connection with different data sources for example to analyze the available data in an environment (horizontal injection). Additionally, it supports to restrict the set of considered topology fragments for the injection by target labels attached to Node Templates to express preferences for the matching [8]. With the OpenTOSCA Injector, concepts for an environment-dependent and automated application deployment can be realized.

Acknowledgments This work was partially funded by the projects SePiA.Pro (01MD16013F), SmartOrchestra (01MD16001F), and IC4F (01MA17008G).

References

1. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing. pp. 692–695. Springer (2013)
2. Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., Wettinger, J.: Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In: International Conference on Cloud Engineering. pp. 87–96. IEEE (2014)
3. Falkenthal, M., Breitenbücher, U., Képes, K., Leymann, F., Zimmermann, M., Christ, M., Neuffer, J., Braun, N., Kempa-Liehr, A.W.: OpenTOSCA for the 4th Industrial Revolution: Automating the Provisioning of Analytics Tools based on Apache Flink. In: Proceedings of the 6th International Conference on the Internet of Things. pp. 179–180. ACM (2016)
4. Hirmer, P., Breitenbücher, U., Binz, T., Leymann, F., et al.: Automatic Topology Completion of TOSCA-based Cloud Applications. In: GI-Jahrestagung, GI, vol. P-251, pp. 247–258. GI (2014)
5. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing. pp. 700–704. Springer (2013)
6. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. OASIS (2013)
7. OASIS: TOSCA Simple Profile in YAML Version 1.0. OASIS (2015)
8. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: Topology Splitting and Matching for Multi-Cloud Deployments. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science. pp. 247–258. SciTePress (2017)
9. Franco da Silva, A.C., Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., Mitschang, B., Steinke, R.: Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017). pp. 358–367. SciTePress (Apr 2017)
10. Franco da Silva, A.C., Breitenbücher, U., Képes, K., Kopp, O., Leymann, F.: OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker. In: Proceedings of the 6th International Conference on the Internet of Things. pp. 181–182. ACM (Nov 2016)