OpenTOSCA – A Runtime for TOSCA-Based Cloud Applications

Tobias Binz[1], Uwe Breitenbücher[1], Florian Haupt[1], Oliver Kopp[1,2], Frank Leymann[1], Alexander Nowak[1], and Sebastian Wagner[1]

[1] IAAS, University of Stuttgart, Germany
[2] IPVS, University of Stuttgart, Germany
firstname.lastname@informatik.uni-stuttgart.de

Abstract TOSCA is a new standard facilitating platform independent description of Cloud applications. OpenTOSCA is a runtime for TOSCA-based Cloud applications. The runtime enables fully automated plan-based deployment and management of applications defined in the OASIS TOSCA packaging format CSAR. This paper outlines the core concepts of TOSCA and provides a system overview on OpenTOSCA by describing its modular and extensible architecture, as well as presenting our prototypical implementation. We demonstrate the use of OpenTOSCA by deploying and instantiating the school management and learning application Moodle.

Keywords: TOSCA, Cloud Applications, Automation, Management, Portability.

1 Background: TOSCA and TOSCA-Based Moodle

The *Topology and Orchestration Specification for Cloud Applications* [4] (TOSCA) is a new OASIS standard to describe Cloud-based applications in a portable and interoperable way. TOSCA standardizes the description of the structure and management aspects (i. e., deployment, operation, termination) of applications. The structure of TOSCA-based applications is defined by a *topology*—a graph of typed nodes and directed typed edges. Nodes represent components forming an application and edges define the relations and dependencies between them. For instance, the topology of the *Moodle* application (`www.moodle.org`) consists of the actual PHP module, an Apache Web Server, a MySQL database, two operating systems (one for the Web server and one for the MySQL database), and two virtual machines (Fig. 1). The relationships in this topology define, for instance, that the Moodle application is "hosted on" a Web server and that the application "connects to" the MySQL database. The types of nodes and relationships specify their properties and management operations. The type "Apache Web Server" defines properties, such as "port" or "version", and management operations, such as "start" or "deploy". The actual implementation of a node is provided by one or many *Deployment Artifacts*, e. g., a Linux VM image, an operating system package for the Apache Web Server, or an archive containing the PHP files of Moodle. In addition, types may define *Implementation Artifacts* that implement the management operations for the respective element. The TOSCA topology and related artifacts are bundled into a *Cloud Service ARchive* (CSAR), the standardized packaging format for TOSCA applications.

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 692–695, 2013.

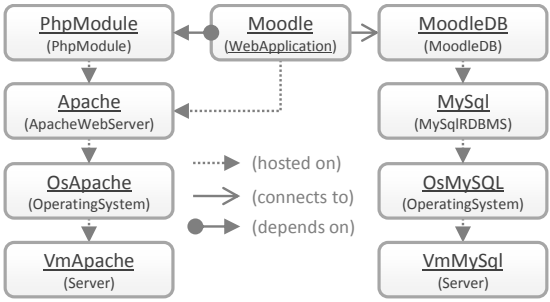

Fig. 1. Moodle Application Topology modeled using *Vino4TOSCA* [2]

TOSCA topologies can be processed by a TOSCA runtime in an imperative or declarative way [5]: *Imperative processing* relies on the implementation of management plans that can be executed fully automated to perform the desired management task, e. g., to instantiate, backup, upgrade, or terminate an application. These high-level management tasks are implemented by orchestrating low-level management operations provided by Implementation Artifacts of nodes and relationships. Because management plans are typically implemented by the application developer, they enable operators to manage the application by running pre-defined plans without the need to understand all the technical details of the management task [1]. Technically, management plans are implemented as workflows. Thus, they inherit properties of workflow technology such as traceability, recoverability, human interaction, and portability. *Declarative processing*, on the other hand, shifts the deployment and management logic from plans to the runtime. To perform the aforementioned high-level management tasks, the runtime has to know the operations that have to be called and their order. Declarative processing is well suited for the deployment of simple applications but is not able to facilitate complex management tasks for various kinds of application structures. For more details, including the TOSCA role model, we recommend the TOSCA specification [4] and TOSCA primer [5].

In summary, TOSCA provides means to describe procedures for managing applications in a standardized way that enable automated and portable processing. With more and more applications described in TOSCA it will enable more and more applications to be hosted in the Cloud.

2 OpenTOSCA: Architecture and Demonstration

OpenTOSCA is a runtime supporting imperative processing of TOSCA applications. Imperative means that the deployment and management logic is provided by plans. The key tasks of OpenTOSCA, addressed by the architecture depicted in Fig. 2, are to operate management operations, run plans, and manage state. Requests to the Container API are passed to the Control component, which orchestrates the different components, tracks their progress, and interprets the TOSCA application. The Core component offers common services to other components, e. g., managing data or validating XML.

Management operations of nodes and relationships are either provided by running (Web) services, e. g., the Amazon EC2 API, or by Implementation Artifacts contained

in the CSAR. In the latter case, the Implementation Artifact Engine is responsible to run these artifacts in order to make them available for plans. Implementation Artifacts, e. g., a SOAP Web service implemented as Java Web archive (WAR), are processed by a corresponding plugin of the engine which knows where and how to run this kind of artifact. The plugins deploy the respective artifacts and return the endpoints of the deployed management operations to be stored in the Endpoints database.

The management plans contained in CSARs are processed by the Plan Engine, which also employs plugins to support different workflow languages, e. g., BPMN or BPEL, and their runtime environments. Plans only define abstractly which kind of service they require but not their concrete endpoints. Therefore, the corresponding plan plugin binds each service invoked by the plan to the endpoint of the management operation before it deploys the plan to the respective workflow runtime. The service's endpoint was added to the endpoint database before by the Implementation Artifact Engine. This way of binding workflows ensures portability of management plans between different environments and runtimes [1]. By using the Plan Portability API, management plans can access the topology and instance information, e. g., the property values of nodes and relationships.

The plugin architecture of the Implementation Artifact Engine and Plan Engine ensure extensibility. Portability is ensured by the two engines working together when binding management plans. Strict separation of architectural components through well-defined OSGi interfaces enables the replacement of implementations of components. This also allows each component to be scaled independently.

Demonstration. In the following, we demonstrate how the OpenTOSCA runtime deploys CSARs and how instances of Cloud applications are created. After uploading the CSAR to OpenTOSCA, the deployment of the TOSCA application follows three steps: (i) First, the CSAR is unpacked and the files are put into the Files store, which is backed either by the local file system or Amazon S3. (ii) Then, the TOSCA XML files are loaded, resolved, validated, and processed by the Control component, which calls the Implementation Artifact Engine and the Plan Engine. The Implementation Artifact Engine deploys the referenced Implementation Artifacts (cf. (a) in Fig. 2) and

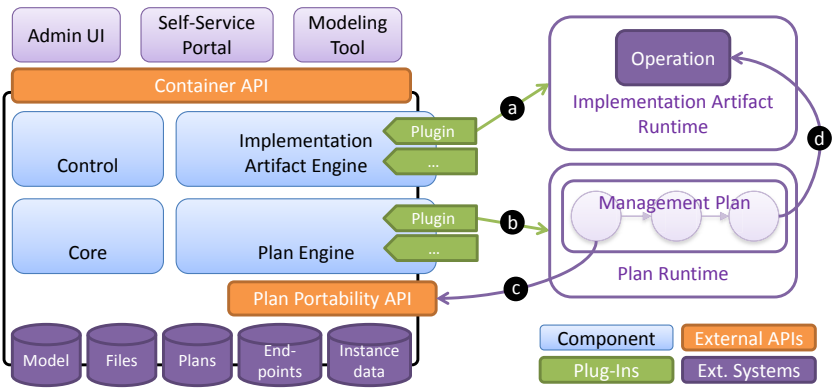

Fig. 2. OpenTOSCA Architecture Overview and Processing Sequence

stores their endpoints in the Endpoints database. (iii) Finally, the Plan Engine binds and deploys the application's management plans (cf. (b) in Fig. 2). The endpoints of the Moodle management plans are stored in the Plans database.

The deployed application can be instantiated by executing the build plan of the application. This plan is either started through the Self-Service Portal, which provides an UI for end user access to the deployed applications, or by sending a SOAP message to it. Credentials (e. g., for Amazon EC2) or configurations (e. g., machine size) are passed as input message to the workflow. The Plan Portability API acts as access point for the plans to the container. By using this API, the topology model, endpoints, and instance data, such as properties of nodes (e. g., the port of a Web server) and relationships, can be read and written (cf. (c) in Fig. 2). Having these data available, the build plan orchestrates the management operations of nodes and relationships to provision and configure the Cloud application (cf. (d) in Fig. 2). To instantiate Moodle, the build plan first starts two virtual machines with a Linux operating system and installs Apache Web Server and MySQL on them. Then, it uses the respective management operations to install the PHP application, import the database schema, and establish the database connection. After completion, a build plan may return certain information, for example, the Web address of the deployed application instance. The Moodle build plan returns the URL of the running Moodle instance, which includes the public URL of the virtual machine running the Apache Web Server. This demonstration is also featured in the *OpenTOSCA demo video* (online at `demo.opentosca.org`).

Currently, OpenTOSCA is used together with the modeling tool "Winery" [3] in the German government-funded projects *CloudCycle* and *Migrate!* as well as in industry and research cooperations of our institute.

Next Steps. To deploy simple applications without the need to model build plans we plan to add declarative processing of applications to OpenTOSCA. We are also working on building a community around OpenTOSCA at `www.opentosca.org`.

Acknowledgments. This work was partially funded by the BMWi projects CloudCycle (01MD11023) and Migrate! (01ME11055). We thank Christian Endres, Matthias Fetzer, Markus Fischer, Nedim Karaoğuz, Kálmán Képes, Rene Trefft, and Michael Zimmermann for their help with the implementation of OpenTOSCA.

References

1. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing 16(3), 80–85 (2012)
2. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In: Meersman, R., et al. (eds.) OTM 2012, Part I. LNCS, vol. 7565, pp. 416–424. Springer, Heidelberg (2012)
3. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – Modeling Tool for TOSCA-based Cloud Applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 702–706. Springer, Heidelberg (2013)
4. OASIS: OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0 Committee Specification 01 (2013)
5. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0 (January 2013)