Portable Cloud Services Using TOSCA

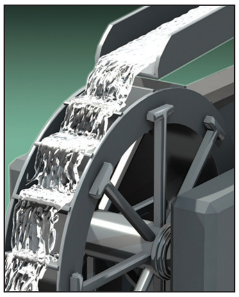
Tobias Binz, Gerd Breiter, Frank Leymann, and Thomas Spatzier

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

BibTeX:

```
@article {ART-2012-09,
    author = {Tobias Binz and Gerd Breiter and Frank Leymann and Thomas Spatzier},
    title = {{Portable Cloud Services Using TOSCA}},
    journal = {IEEE Internet Computing},
    address = {Los Alamitos, CA, USA},
    publisher = {IEEE Computer Society},
    volume = {16},
    number = {03},
    pages = {80--85},
    month = {May},
    year = {2012},
    doi = {http://doi.ieeecomputersociety.org/10.1109/MIC.2012.43},
    issn = {1089-7801}
 }
```

Editor: Schahram Dustdar • dustdar@infosys.tuwien.ac.at



Portable Cloud Services Using TOSCA

Tobias Binz • *University of Stuttgart*

Gerd Breiter • *IBM Boeblingen Laboratory*

Frank Leymann • *University of Stuttgart*

Thomas Spatzier • *IBM Boeblingen Laboratory*

For cloud services to be portable, their management must also be portable to the targeted environment, as must the application components themselves. Here, the authors show how plans in the Topology and Orchestration Specification for Cloud Applications (TOSCA) can enable portability of these operational aspects.

Underneath all the hype, the essence of cloud computing is the industrialization of IT. Similar to mass production lines in other industries (such as the auto industry), cloud computing standardizes offered services and thus increases automation significantly. Consequently, enterprises are increasingly utilizing cloud technology; however, major challenges such as portability, standardization of service definitions, and security remain inadequately addressed. The ability to move cloud services and their components between providers ensures an adequate and cost-efficient IT environment and avoids vendor lock-in.

Research has already addressed movability and migration on a functional level.[1,2] However, no one has yet examined cloud service portability with regard to management and operational tasks, which are a significant and increasing cost factor. One reason is the lack of an industry standard for defining composite applications and their management. Without an appropriate standardized format, ensuring compliance, trust, and security — the biggest area of critique preventing the cloud's wider adoption — is difficult. Dealing with these challenges in industry and research has the potential to bring cloud computing to the next level.

Here, we present how the portable and standardized management of cloud services is enabled through the Topology and Orchestration Specification for Cloud Applications (TOSCA),[3] a recently initiated standardization effort from OASIS. We show how TOSCA *plans* — which capture the management aspects of cloud services in a reusable way — use existing workflow technologies and research results to facilitate the portable, automated, and reusable management of cloud services throughout their life cycle.

Cloud Services Life Cycle

Before examining the cloud services life cycle (see Figure A in the Web appendix at http://doi.ieeecomputersociety.org/10.1109/MIC.2012.43), let's look at the fundamental roles and concepts involved. IBM's Cloud Computing Reference Architecture (CCRA)[4] defines three main roles typically encountered in any cloud computing environment: the *cloud service creator*, *cloud service provider*, and *cloud service consumer*. Each role can be fulfilled by a single person, a group of people, or an organization. Following the CCRA definitions, the service creator develops cloud services; the service provider runs those services, and provides them to service consumers, which can also be IT systems. Consumers can be billed for all (or a subset of) their interactions with cloud services and the provisioned service instances.

According to the CCRA definition, a cloud service represents any type of (IT) capability that the service provider offers to service consumers. In contrast to traditional IT services, cloud services have attributes that are typically associated with cloud computing, such as a pay-per-use model, self-service usage, scalability, and resource sharing.

Let's now examine a cloud service's life cycle, as defined in prior work.[5] In the *definition* phase, all management knowledge required for the specific cloud service — in particular, how to instantiate it — is captured in a *service template*. This knowledge includes information about the cloud service's internal structure (that is, its *topology*) along with operational and management aspects to build, manage, and terminate cloud services.

In the *offering* phase, the cloud service provider creates a *cloud service offering* based on a service template, adding all provider- and offering-specific information. This includes aspects such as pricing and specific technical information such as IP address range and application configurations. Finally, the offering is published in a *service catalog*.

In the *subscription and instantiation* phase, the cloud service consumer browsing the service catalog can select and subscribe to the respective offering. The consumer customizes the service through points of variability (for example, selecting "small," "medium," or "large" for the service's size), signs a contract, and accepts the offering's terms and conditions. This subscription process triggers the instantiation of the cloud service instance. The cloud management platform aggregates all the required resources from the common resource pools, for example infrastructure components, and automatically deploys, installs, and configures the service's necessary pieces.

In the life cycle's *production* phase, the cloud management platform uses management plans to manage the service instance for compliance with the service-level agreements (SLAs) negotiated at subscription time. For example, the management platform assigns additional resources to the instance when the number of users increases, and removes them when users are no longer using the service. The cloud service provider or consumer can also trigger management plans manually — for example, to back up or upgrade the service.

Finally, when the cloud service consumer decides to get rid of the service or the subscription expires, the service instance terminates, and all the resources go back into the resource pool.

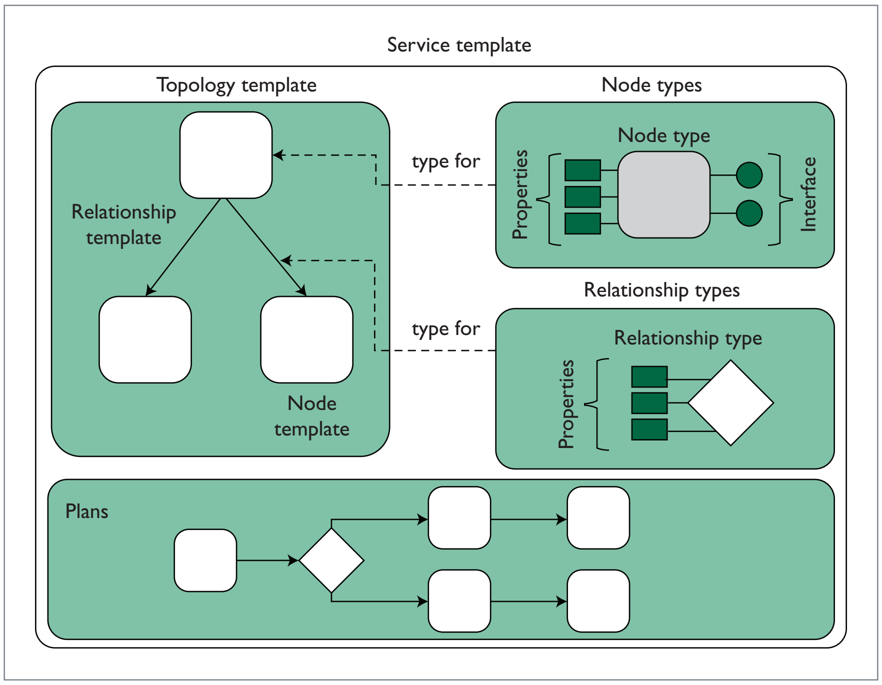


Figure 1. The TOSCA service template. Nodes represent the service's components, whereas relationships connect and structure nodes into the topology. Plans capture the operational aspects of the cloud service. (Figure adapted from the OASIS Topology and Orchestration Specification for Cloud Applications.[3])

Topology and Orchestration Specification

TOSCA describes composite applications and their management in a modular and portable fashion. It thus defines service templates that contain a cloud service's topology (for instance, an application is hosted on an application server, which is in turn hosted on an operating system) and its operational aspects (such as how to deploy, terminate, and manage this service). Service templates are interpreted by a TOSCA-compliant environment, which operates the cloud services and manages their instances. Figure 1 depicts a service template's main elements.

The creator of a cloud service captures its structure in a service topology — a graph with nodes and relationships. Nodes represent the service's components, and relationships connect and structure nodes into the topology. Both nodes and relationships are typed and hold a set of type-specific properties, bringing

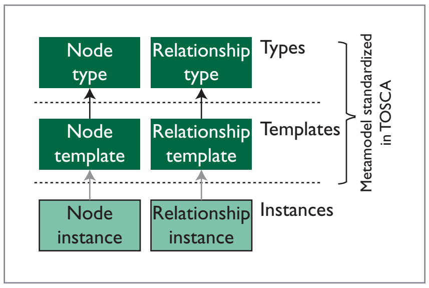


Figure 2. Nodes and relationships in TOSCA. Types define reusable entities and their properties, and templates form the cloud service's topology using these types, which are then instantiated as instances of the described cloud service.

meaning and variability to these generic TOSCA elements. We position these elements into three levels: *types* define reusable entities and their properties, *templates* form the cloud service's topology using these types, which are then instantiated as *instances* of the described cloud service (see Figure 2). TOSCA specifies the metamodel for types and templates — that is, the language for defining them. Concrete types aren't part of the specification and must be standardized by the respective domain expert groups. Furthermore, TOSCA is independent of concrete providers and defines a model to describe all kinds of services and their ingredients.

Nodes in particular define a range of information to deploy, terminate, and manage the cloud service: *instance states* represent the node's internal state during production as part of the topology. *Deployment artifacts* represent the artifacts needed to actually deploy the defined service — that is, application files providing the service's functionality. One key to support interoperability and reusability is that nodes expose their management operations explicitly as part of the topology. Moreover, nodes can provide the implementations of these management operations as *implementation artifacts*. This enables plans to orchestrate the node's fine-grained management functionalities into higher-level management functionalities of the service, as we discuss next.

The Concept of Plans

Managing services requires extensive, mostly manual effort by the customers. Each organization using an application learns on its own how to operate that application, acquires management knowledge, and automates certain aspects in scripts. TOSCA enables application developers and operators to model management best practices and reoccurring tasks explicitly into so-called plans. If these plans were portable between different environments and providers, the achieved reusability and automation of service management would significantly reduce the total cost of ownership. Everyone using the service could benefit from the accumulated knowledge by executing the respective plan without needing to understand all the technical details. In particular, self-service and rapid elasticity, two important cloud service properties, can be realized only if service management is automated to a large degree. Figure 3 depicts a (simplified) management plan that sets up an IBM WebSphere cluster. We next look at plans in detail and illustrate why TOSCA plans are portable.

Plans Are Workflows

Instead of introducing a new way to define workflows, plans use existing workflow languages such as Business Process Model and Notation (BPMN)[6] or the Business Process Execution Language (BPEL).[7] Moreover, plans can use any workflow language supported by a management environment. There are many reasons to introduce workflow technology into service management. In the manual or scripting-based approaches used today, managing data and properly handling errors is a pain, especially if the tasks are frequently executed or long-running. In addition to plans' ability to explicitly define actions to handle exceptional cases, compensation is a powerful concept. In contrast to databases, which are able to roll back changes, plans — which initiate changes in the physical world and external systems — often can't roll back what they've done. Thus, plans realize compensation by explicitly modeling for each activity how to undo it, in the same language the plan is written. For example, if a plan starts a server, a straightforward compensation would be to shut it down. Much more complex is the compensation of a software upgrade, which might require going back to a certain snapshot of the system. In automated cloud environments,

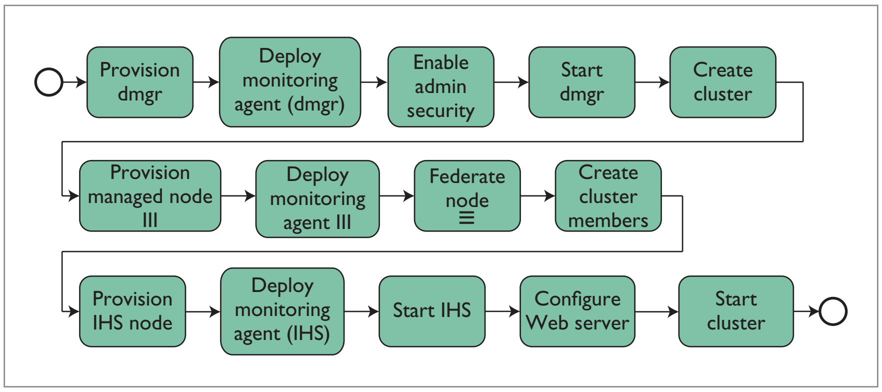


Figure 3. Simplified plan for the initial provisioning of a WebSphere cluster. The plan exemplarily depicts the steps required to facilitate this task.

error handling is crucial because immediate human intervention isn't possible. Furthermore, using workflow technology ties people into a plan. This enables manual interactions required during cloud services management, such as determining and passing parameters to management actions or kicking off manual activities.

How Plans Benefit from the Topology Model

Plans use the cloud service topology in three ways: First, plans orchestrate the management interfaces and operations defined in TOSCA nodes. Operations are described in the Web Services Description Language (WSDL), Representational State Transfer (REST), or scripts that implement particular management operations on the respective node. These operations might be external services, or their implementation might be included in the service template as an implementation artifact. In the latter case, the management environment ensures that these implementations are in place before the service template is instantiated.

Second, plans can inspect the topology model to access a service's nodes and relationships. This enables flexible plans whose behavior is based on the topology and changes therein. For example, a plan installing an operating system patch could iterate over the topology to dynamically find the respective nodes.

Finally, plans read and write a service's instance information — that is, the node's instance state, such as properties containing credentials, IP addresses, and so on. The workflow engine manages the state inside a plan and delivers it to the different activities. Between plans, the management environment stores instance information externally. Plans must ensure that changes they did to the service instance are reflected in the instance information so other plans can use them.

Figure 4 provides an overview of these concepts. The plan first retrieves information about the topology and the respective service instance, namely deployment artifacts, properties, state, policies, and so on. The plan then processes this data and invokes an operation that the node provides. Additionally, the plan can invoke external Web services and APIs using the standard mechanisms of the workflow language, as when using a cloud provider's offering. Finally, the plan updates the node's instance information to reflect the changes made. This example shows how the different concepts are connected through plans. Note that a plan can modify any number of nodes — that is, it can execute management operations on a set of nodes or the whole topology.

If enough semantic information is included in the topology, some types of plans can be generated automatically. This shifts the burden from the plan to the topology modeler to enrich the topology with sufficient semantic information. TOSCA supports both approaches, and compliant management environments can interpret the resulting service template.

What Makes Plans Portable?

TOSCA plans' portability is inherited from the workflow language and engines used. With the exception of extension activities, workflow languages such as BPMN and BPEL are portable between different engines. Workflow portability isn't enough because the orchestrated services must also be available. In TOSCA, these services are explicitly described in the nodes as operations. For operations referencing external services, portability isn't a problem. If the operation represents a service that's delivered as part of the service template (as an implementation artifact), TOSCA uses an approach similar to plans. Implementation artifacts are implemented in standardized

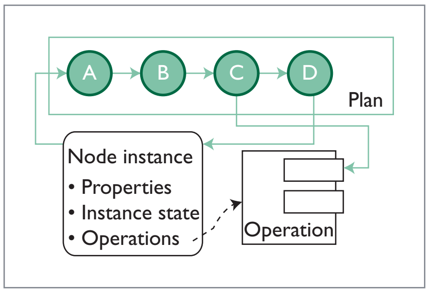


Figure 4. Relationship between plans and the service topology. The plan retrieves information from the node instance, processes it using operations, and updates the data stored for the node instance.

languages, such as Java, Perl, PHP, and shell scripts. For these languages, runtimes are available for different environments. Either the management environment or the cloud service provider deploys them before the cloud management platform instantiates the service template for the first time.

Thus, TOSCA can operate a service template in management environments that support the required plan and implementation artifact languages. To increase portability, TOSCA enables a cloud service creator to provide the same plan or implementation artifact in different languages. For example, a cloud service creator can include a plan with the same functionality twice: in BPEL and BPMN. A transformation into another language might even be possible automatically. This increases the number of management environments in which the service template can operate.

TOSCA allows service creators to gather into plans those activities necessary to deploy, manage, and terminate the described cloud service. Due to the portability of plans and the operations they use, TOSCA services can be operated in different management environments. However, these environments must support the set of workflow languages and implementation artifacts included in the service template. Furthermore, access to the topology model and instance information management should be provided by the management platform.

As part of the CloudCycle project (www.cloudcycle.org/en), funded by the German Federal Ministry of Economics and Technology, we're implementing a prototypical, open source TOSCA management environment. To address the third major challenge of cloud computing we identified in the introduction, we're researching how to ensure compliance and security throughout the entire service life cycle. Security experts are enabled to annotate policies to TOSCA elements. Policy-specific plug-ins will then enforce these policies in the management environment during the cloud service's whole life cycle. TOSCA provides an exchange format for policies but no guarantee that the management environment and providers also enforce these policies. This must be ensured by a trustworthy entity through certification of management environments, providers, and services.

TOSCA models contain myriad references to files, such as those with additional TOSCA definitions, plans, and other artifacts. Consequently, future work must be invested toward a TOSCA archive, a self-contained package of these artifacts, similar to Java EAR files. This cloud service could then be offered in the marketplace and, due to its portability, operated in different management environments.

References

1. F. Leymann et al., "Moving Applications to the Cloud: An Approach Based on Application Model Enrichment," *Int'l J. Cooperative Information Systems*, vol. 20, no. 3, 2011, pp. 307–356.
2. T. Binz, F. Leymann, and D. Schumm "CMotion: A Framework for Migration of Applications into and between Clouds," *Proc. Int'l Conf. Service-Oriented Computing and Applications*, IEEE Press, 2012, pp. 1–4.
3. *Topology and Orchestration Specification for Cloud Applications (TOSCA)*, OASIS specification, Oct. 2011; www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.
4. M. Behrendt et al., "IBM Cloud Computing Reference Architecture," Open Group submission, Feb. 2011; www.opengroup.org/cloudcomputing/uploads/40/23840/CCRA.IBMSubmission.02282011.doc.
5. G. Breiter and M. Behrendt, "Lifecycle and Characteristics of Services in the World of Cloud Computing," *IBM J. Research and Development*, vol. 53, no. 4, 2009, pp. 3:1–3:8.
6. *Business Process Model and Notation (BPMN) Version 2.0*, Object Management Group specification, Jan. 2011.
7. *Web Services Business Process Execution Language (BPEL) Version 2.0.*, OASIS specification, 2007.

Tobias Binz is a researcher and coordinator of the CloudCycle project in the Institute of Architecture of Application Systems at the University of Stuttgart, Germany. His research focuses on the migration of applications to cloud environments while ensuring their security and compliance. Binz is an architect for OpenTOSCA, an open source cloud service management platform supporting the Topology and Orchestration Specification for Cloud Applications (TOSCA). Contact him at tobias.binz@iaas.uni-stuttgart.de.

Gerd Breiter is an IBM Distinguished Engineer and Tivoli Chief Architect for Cloud Computing in the IBM Research and Development Laboratory, Boeblingen, Germany, where he's one of the key technicians defining IBM's cloud computing architecture and strategy. His research areas include utility, on-demand, and cloud computing, as well as the architecture for the build-out of hybrid clouds. Breiter coleads the definition of the IBM Cloud Computing Reference Architecture (CCRA). Contact him at gbreiter@de.ibm.com.

Frank Leymann is a professor of computer science and the director in the Institute of Architecture of Application Systems at the University of Stuttgart, Germany. His current research focuses on many aspects of cloud, service, and business-process technology. Formerly at IBM, Leymann was chief architect of IBM's workflow/process management technology, coleader of the Web Services Architecture team, and architect of IBM's Service Bus. Contact him at frank.leymann@iaas.uni-stuttgart.de.

Thomas Spatzier works at the IBM Research and Development Lab in Boeblingen, Germany. His current research includes architecture and design in IBM's cloud management product family. Spatzier is actively involved in cloud computing standardization activities, including as a co-editor of the OASIS TOSCA specification. Contact him at thomas.spatzier@de.ibm.com.