Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing

Tim Waizenegger[1], Matthias Wieland[1], Tobias Binz[2], Uwe Breitenbücher[2], Florian Haupt[2], Oliver Kopp[1,2], Frank Leymann[2], Bernhard Mitschang[1], Alexander Nowak[2] and Sebastian Wagner[2]

[1] Institute for Parallel and Distributed Systems, University of Stuttgart, Germany
{firstname.lastname}@ipvs.uni-stuttgart.de
[2] Institute of Architecture of Application Systems, University of Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings {INPROC-2013-43,
    author = {Tim Waizenegger and Matthias Wieland and Tobias Binz and
              Uwe Breitenb{\"u}cher and Florian Haupt and Oliver Kopp and
              Frank Leymann and Bernhard Mitschang and Alexander Nowak
              and Sebastian Wagner},
    title = {{Policy4TOSCA}: A Policy-Aware Cloud Service Provisioning
             Approach to Enable Secure Cloud Computing},
    booktitle = {On the Move to Meaningful Internet Systems: OTM 2013
                 Conferences},
    year = {2013},
    publisher = {Springer Berlin Heidelberg},
    isbn = {978-3-642-41029-1},
    doi = {10.1007/978-3-642-41030-7_26}
 }
```


The original publication is available at
http://dx.doi.org/10.1007/978-3-642-41030-7_9
See also LNCS-Homepage:
http://www.springeronline.com/lncs

Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing

Tim Waizenegger[1], Matthias Wieland[1], Tobias Binz[2], Uwe Breitenbücher[2], Florian Haupt[2], Oliver Kopp[1], Frank Leymann[2], Bernhard Mitschang[1], Alexander Nowak[2], and Sebastian Wagner[2]

[1] Institute of Parallel and Distributed Systems
[2] Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstr. 38
70569 Stuttgart, Germany
{firstname.lastname}@informatik.uni-stuttgart.de

Abstract. With the growing adoption of Cloud Computing, automated deployment and provisioning systems for Cloud applications are becoming more prevalent. They help to reduce the onboarding costs for new customers as well as the financial impact of managing Cloud services by automating these previously manual tasks. With the widespread use of such systems, the adoption of a common standard for describing Cloud applications will provide a crucial advantage by enabling reusable and portable applications. TOSCA, a newly published standard by OASIS with broad industry participation provides this opportunity. Besides the technical requirements of running and managing applications in the cloud, non-functional requirements, like cost, security, and environmental issues, are of special importance when moving towards the automated provisioning and management of Cloud applications. In this paper we demonstrate how non-functional requirements are defined in TOSCA using policies. We propose a mechanism for automatic processing of these formal policy definitions in a TOSCA runtime environment that we have developed based on the proposed architecture of the TOSCA primer. In order to evaluate our approach, we present prototypical implementations of security policies for encrypting databases and for limiting the geographical location of the Cloud servers. We demonstrate how our runtime environment is ensuring these policies and show how they affect the deployment of the application.

Keywords: Cloud Computing, TOSCA, Cloud Service, Cloud Management, Policy-Framework, Security, Green-IT, Sustainable Cloud Service

1 Introduction

In recent years, the steadily increasing use of IT in enterprises has lead to higher management efforts concerning the whole application lifecycle. This becomes a

challenge for enterprises as the degree of complexity increases with each new application and technology [5], whilst manual operator errors account for the largest fraction of failures [17].

Cloud computing tackles these issues by enabling enterprises to automate and outsource their IT, as well as its management [7]. Therefore, automated deployment and provisioning systems for Cloud-based applications become more and more important and prevalent. They help to reduce the onboarding costs for new customers as well as the financial impact of managing Cloud services by automating these previously manual tasks. With the widespread use of such systems, the adoption of a common standard for describing C applications that prevent vendor lock-in. Therefore, the Topology and Orchestration Specification for Cloud Applications (TOSCA [14]) was published as a new Cloud standard by OASIS in 2013 with broad industry participation.

TOSCA enables application developers to model and package their Cloud applications including all management aspects in a portable, interoperable, and standardized fashion. This allows automating the whole application lifecycle management from provisioning, over maintenance, to termination. In addition, applications can be deployed on various kinds of infrastructures, integrated and combined with any XaaS offerings, and even migrated between different Cloud providers. As non-functional requirements, like cost, security, and environmental issues, are very important when providing, using and managing applications, TOSCA allows specifying policies that express such non-functional requirements. However, TOSCA lacks a detailed description of how to apply, design, and implement policies.

In this paper, we tackle this issue and demonstrate how non-functional requirements can be defined in TOSCA as policies. We exemplarily use policies emerging from the security domain. We further demonstrate how a prototypical TOSCA runtime environment processes those policies. We propose two mechanisms for implementing policy-specific logic: (i) Policy-aware Management Plans and (ii) Policy-aware Management Operations. The presented approach enables Cloud providers as well as application developers to specify, design, and implement various kinds of policies.

The remainder of this paper is structured as follows: Section 2 describes related work in the field of Cloud service deployment. Section 3 provides an introduction into the TOSCA standard and briefly introduces our prototypical TOSCA runtime implementation. Section 4 explains the formal definition of policies in the TOSCA standard and introduces our example applications and policies. Section 5 presents the automated processing of policies in TOSCA following two approaches including a discussion. Finally, Section 6 concludes the paper.

2 Related Work

NIST [10] defines Cloud computing as a model for enabling ubiquitous, convenient, and on-demand network access to a shared pool of configurable computing

resources. An important aspect of Cloud Computing is the fast deployment of the resources with minimal management effort. To achieve this goal, the resources have to be formally described in a so-called *Cloud service*. A Cloud service is a *composite application* that consists of different components. The setup of this Cloud service is defined in an *application model* which defines the *topology* of the service.

There are different approaches available that enable the description of composite applications. Unger et al. [21] present an application model, which allows describing dependency and deployment relations between components. Cafe [11] is a system that supports the modeling and deployment of composite applications. It is based on a formal definition of an application model. Leymann et al. [8] broadens this model and adds labels to optimize the distribution of applications between different Clouds.

Furthermore, there are systems available in the industry to describe application models, like the Service Component Architecture (SCA) [1]. SCA allows composing applications out of services by defining functional relations. Hence, other relations, e.g., where a service is deployed, are not captured. In software architecture, design, and development languages are used to describe the structure of applications. There are different languages available, e.g., Acme architectural description language [6] or the well-known Unified Modeling Language (UML) [16]. For instance, Machiraju [9] uses UML to model application structures in the scope of topology discovery. However, they target mostly application architectures and do not allow to model formal policy-aware service topologies, as it is needed in this paper.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) is a standardization initiative by OASIS to define the topology and management aspects of Cloud applications. This paper is using the concepts of TOSCA, because TOSCA allows describing the Cloud service topologies, the deployment process of the services and the policies that have to be followed. Thus, all aspects we need for Cloud service provisioning can be specified in TOSCA.

The focus of the previously published paper [22] proposing a policy-framework for the deployment and management of Cloud services is the definition of a taxonomy for policies and their aspects. Based on that a signature of a policy can be defined. Furthermore, a high-level architecture of a policy-framework is presented. The current paper describes a technical solution for the deployment stage in the lifecycle aspect of a policy defined by Waizenegger et al. [22]. Furthermore, the taxonomy defined by Waizenegger et al. is used for the formal definition of policies in this paper.

The goal of this paper is to introduce different aspects of policies and to evaluate how to use policies for Cloud service deployment and management with TOSCA. One use case for policy-aware Cloud service provisioning is security. A good way to achieve security in Cloud Computing is certification [19]. Certification is only possible if the Cloud service description is not changed after it was certified, so different offerings for one Cloud service description may be available. An offering defines which policies of the set of possible policies defined in the Cloud

service description should be enforced when the Cloud service is being deployed. A Cloud service description can be certified, but customers may still select the needed configuration based on choosing a proper offering. There are publications that focus on specifying frameworks for implementing different methods to provide security features, e.g., authentication across different providers or trust management [20].

In an earlier publication [4], we describe a framework that enables the fully automated provisioning of applications under compliance with policies that are used to configure, guide, and extend the provisioning. However, this approach is limited as polices can be attached only to nodes and relations, not to the overall Topology Template. This restricts application developers from defining high-level requirements such as all-encompassing data-location policies that apply to the whole application. This limitation forces them to use fine-grained policies on individual elements, which is not sufficient in many cases as discussed in Sect. 5.

The approach described by Nowak et al. [13] allows to build TOSCA service descriptions based on a set of different service description patterns. However, these service descriptions patterns are defined in a generic manner. Thus, using policies as described in this work would enable the configuration of those service descriptions regarding concrete use cases.

3 Introduction to TOSCA

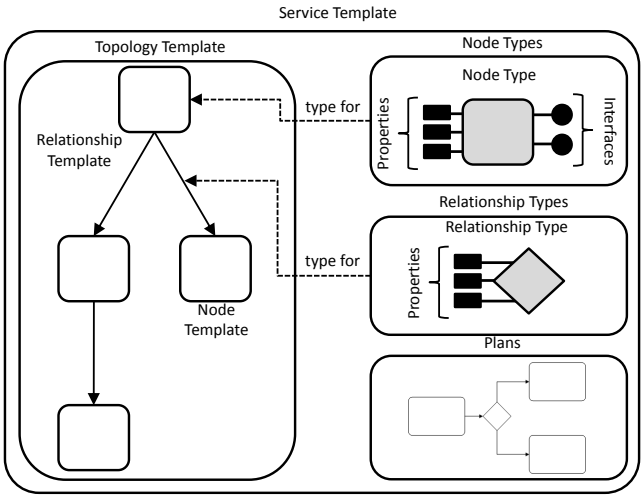

Fig. 1. TOSCA Service Template

A Cloud service in TOSCA is defined by two parts, first, the topology defining the structure of the application and, second, the orchestration part which defines

the deployment and management of the applications components. In the TOSCA jargon, an application model comprised of the above mentioned elements is called a *Service Template* (see Fig. 1).

The topology is a graph of typed nodes and directed, typed edges called a *Topology Template*. The types, which can be extended and derived, define, for example, the properties and management operations of the respective nodes and edges. The node type *Apache Webserver*, for example, has properties like "port" or "version" and management operations like "start webserver" or "deploy an application".

Either management operations are shipped as part of the TOSCA application, called *Implementation Artifacts*, or they are realized by an external service like the *Amazon EC2 Management Web Service*, or a combination of both. Types are instantiated in the topology by so-called *Node Templates* or *Relationship Templates*, respectively. Templates define the actual properties and how the components of the application are connected. The actual implementation of the nodes is provided as a *Deployment Artifact* in the templates, for example, a ZIP file containing the Apache Webserver or a WAR file embodying the Web service node.

Figure 2 shows an example TOSCA application topology. The topology shows a LAMP-based application (Linux, Apache, MySQL, and PHP) which runs on two virtual machines hosted on Amazon EC2. The rounded rectangles depict node templates, the vertical arrows between the node templates depict relationship templates of type "hosted-on". As we use Vino4TOSCA [3] to visualize application topologies, all types are depicted as text enclosed by parentheses.

TOSCA addresses the automation and portability of the application's management aspects through Management Plans. Management Plans are based on existing workflow technology and orchestrate the type-specific management operations of the nodes and edges into higher-level application-specific management operations [2]. A special *Build Plan* defines how to instantiate the service. Refering to our example topology in Fig. 2 the Build Plan first starts the two Ubuntu virtual machine instances on Amazon EC2, then installs the Apache Webserver and MySQL RDBMS, creates a new database, and in the last step imports the database schema, installs the Java application, and configures the database connectivity in the Java application. Another Management Plan for the backup of the example application may create a dump of the database and archive this file.

All the artifacts required for the application are packaged into a standardized *Cloud Service ARchive* (CSAR) together with the actual TOSCA files, serialized in XML. Following the TOSCA principles, the CSAR file is portable between different TOSCA runtimes. The TOSCA runtime is responsible for running the Implementation Artifacts and Management Plans, provide the artifacts contained in the CSAR file, and hold the state of the application. A generic architecture of an imperative TOSCA runtime is described by OASIS [15]. Here, *imperative* means that plans provide the deployment and management logic, and not the runtime itself.

On a high-level, the CSAR file is processed as follows: (1) The TOSCA definitions are validated and analyzed. (2) Then the Implementation Artifacts are deployed by the TOSCA runtime into a suitable container for the respective Implementation Artifacts. Our TOSCA runtime consits of a container for executing the plans (BPEL engine) as well as another container that hosts the Implemenation Artifacts (Apache Tomcat). (3) Afterwards, the Management Plans, which use the operations provided by the Implementation Artifacts, are bound and deployed to a process engine. (4) Now it is possible to deploy and manage the application provided by the CSAR file.

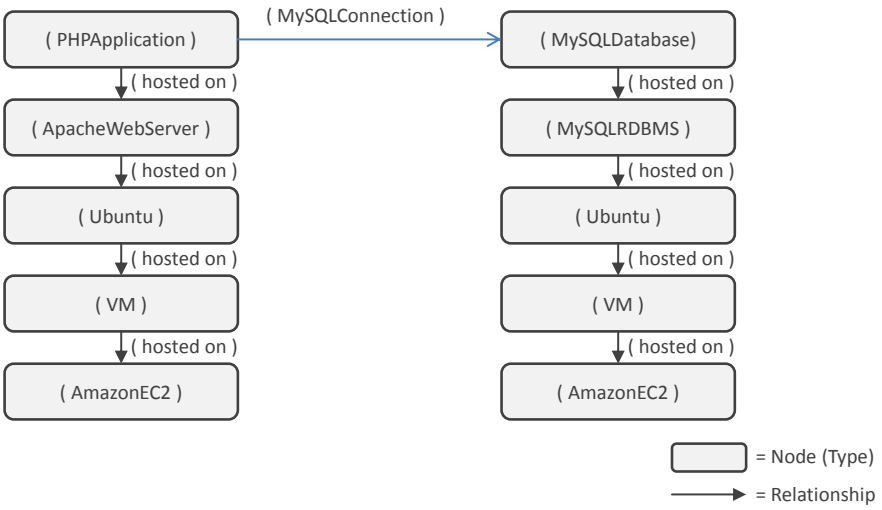

Fig. 2. Topology Template of a LAMP-based TOSCA Application

Similar to the other entities in the TOSCA standard, a policy has an abstract definition; a *Policy Type* (see Listing 1.1). It is instantiated by defining a *Policy Template* (see Listing 1.2). While the Policy Type describes the structure and required parameters of a policy, the Policy Template is used to define a specific policy instance that is annotated to an entity of the topology. A Service Template then contains the topology comprised of Node and Relationship Templates as well the Policy Templates that define which policies are in effect for which entities of the topology. The generic type definitions for relationships, nodes and policies are also contained in the Service Template since they are required as definitions for the specified templates.

4 Policy Fundamentals and Formal Definitions

In order to make automatically deployable Cloud services secure and still keep the application model flexible, we propose the use of policies that formalize non-

```
<PolicyType name="xs:NCName"
            policyLanguage="xs:anyURI"?
            abstract="yes|no"?
            final="yes|no"?
            targetNamespace="xs:anyURI"?>
  <Tags> ?
    <Tag name="xs:string" value="xs:string"/> +
  </Tags>

  <DerivedFrom typeRef="xs:QName"/> ?
  <PropertiesDefinition element="xs:QName"? type="xs:QName"?/> ?

  <AppliesTo>
    <NodeTypeReference typeRef="xs:QName"/> +
  </AppliesTo> ?

  policy type specific content ?
</PolicyType>
```

Listing 1.1. Policy Type definition according to TOSCA

```
<PolicyTemplate id="xs:ID" name="xs:string"? type="xs:QName">
  <Properties>
    XML fragment
  </Properties> ?

  <PropertyConstraints>
    <PropertyConstraint property="xs:string"
                        constraintType="xs:anyURI"> +
      constraint ?
    </PropertyConstraint>
  </PropertyConstraints> ?

  policy type specific content ?
</PolicyTemplate>
```

Listing 1.2. Policy Template definition according to TOSCA

functional security requirements. In this paper, we present two mechanisms for processing policies during the deployment and management of Cloud services in TOSCA and demonstrate how they affect these processes using example policies and a Service Template. These example policies and application are introduced at the end of this Section.

In the following, we present a taxonomy for classifying and describing Cloud service policies and show how policies, application models and offerings interact.

Considering the reusability of application models and the involvement of multiple parties like service providers, developers and Cloud providers, the defined policies must follow a common standard that describes what their effect is. This way the suitability of a certain Service Template can be evaluated, and multiple implementations can be compared. Therefore, we propose a taxonomy for the description of policies that was introduced by Waizenegger et al. [22]. In the following, we outline this taxonomy and introduce the syntax and semantics of formal policy definitions in TOSCA.

As an abstract definition, we see a Cloud service policy as a tuple of elements that describe its behavior and effect. This tuple is given in Definition (1).

$$\text{policy} = \langle \text{stage}, \text{layer}, \text{effect}, \text{property} \rangle \tag{1}$$

We define our taxonomy based on the values of the elements of this tuple. They allow the categorization and comparison of policies in different service templates but are not meant to provide the means for automatically deriving the policy implementation.

The first three elements identify any given policy while the **property** is used to specify the behavior of the policy. Depending on the requirements of the specific policy, this element can be a complex type, an atomic value or it can be absent if the policy is sufficiently described by the other elements. The following will give a brief description of these parameters.

Stage: Identifies the stage in the lifecycle of the Cloud service at which the policy is being applied. The stages include solution building, instantiation, runtime and termination. It should be noted that the policy might still affect other stages than the one it was applied to.

Layer: The layer defines which part of the topology the policy applies to. It allows for the definition of localized policies that apply to individual nodes and relationships as well as global policies that apply to the entire topology. A discussion of global and localized policies is given in Section 5.

Effect: This element gives a description of what the effect of the policy is i.e. what purpose it has or what aspect of the service it modifies.

In the following, two example policies are given that we use as an application for our taxnomomy and the policy processing methods presented in the remainder of this paper. The first policy, called the *Region Policy*, determines the geographical location of the data center in which the service is deployed. This is necessary for services that are legally obligated to store and process their data in a certain jurisdiction. Since this policy determines an aspect of the underlying infrastructure, it is tightly coupled with the Cloud provider. For any given (virtual) machine, it is a non-trivial problem to determine its physical location with virtual means alone. Therefore, we use an API offered by the Cloud provider to advise them to use a data center in a certain region.

With our second policy (*Database Encryption Policy*), we enable database encryption since this is a frequent customer requirement. This policy does not require interaction with the Cloud provider as it is a purely internal setting of the database component used in the Service Template. It serves as an example to show how policies affect the setup and configuration of service components.

Using the taxonomy introduced above, the values for the Region Policy are as follows. *Stage:* instantiation, *layer:* virtual machines, *effect:* determine the geographical location. The *property* now indicates the specific geographical region chosen for the service, e.g., EU.

The Database Encryption policy is described as follows. *Stage:* instantiation, *layer:* database node, *effect:* enable encypted datastore, *property:* AES256.

In order to apply a single, or multiple policies to a Service Template, an offering as given by Definition (2) is constructed.

$$\text{offering} = \langle \text{service\_template}, \{\text{policies}\} \rangle \tag{2}$$

It is comprised of the Service Template itself as well as a list of policies that govern the lifecycle of the service described by this offering. The process of selecting an offering is given in Fig. 5.

The function *instantiate* represents the service deployment process and is given in Definition (3).

$$\text{instantiate}(\text{offering}, \text{service\_parameters}) \tag{3}$$

It creates an actual instance of the Service according to the given list of policies. This process is implemented by the TOSCA runtime which interpretes the Service Template and applies the policies contained in the offering. Additional *service parameters* are passed to this function to represent the generic, non-policy related information required by the deployment process. A detailed description of this process in given by Section 5 and referring Fig. 6.

In order to provide an application for our policies and to demonstrate their processing during service instatiation, we created a Service Template for the Web-based learning management system Moodle. The topology of our Service Template is given in Fig. 3. Moodle is a prime candidate for our two policies introduced above since it deals with sensitive personal information that is often required to be processed and stored in a certain jurisdiction. Database encryption therefore is also often a requirement in Moodle installations.

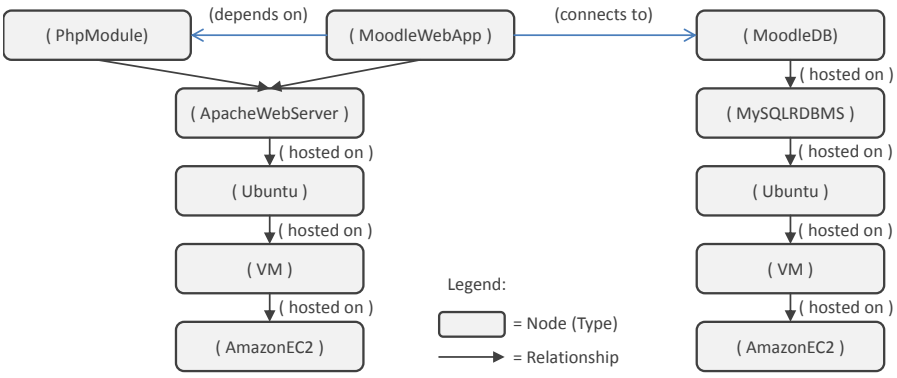

Fig. 3. Moodle Application Topology using Vino4TOSCA [3]

5 Methods for Policy-Aware Cloud Service Provisioning

As far as the TOSCA standard goes, only the annotation of policies in the Service Template is standardized as described above. This leaves various aspects of policy definition and processing up to interpretation and could lead to the emergence of different incompatible implementations that are all standard compliant. We therefore had to make certain assumptions and decided on a concept for processing policies that is in line with the idea of the TOSCA standard.

Enabling and Configuring Policies through Offerings: Once a policy is annotated in the Topology Template, we consider this policy supported by the solution package. There still exists the need to enable or disable certain policies as well as allowing their configuration by providing the property described in Sect. 4. The TOSCA standard does not yet cover any of those requirements, so two approaches to this issue are possible.

First, all policy properties could be defined in the annotation and then read by the plans. This would require modifying the Service Template in order to configure the policies to the customer's specific needs, prohibiting the use of generic application models and making it impossible to validate and digitally sign Service Templates.

Therefore, we chose a second approach in which configuration parameters for the annotated policies are given by an *Offering*. This is implemented by providing different instances (Policy Templates) of a Policy Type that are each part of an offering. This way it is possible to choose a specific set of policies to be active from a single CSAR file without requiring to modify the Service Template. This is especially useful in scenarios with high numbers of customers and policies that require individual configuration.

Interfering Policies: Since it is the purpose of policies to modify the structure and behavior of the service, the possibility exists that certain policies interfere with each other either by affecting the same aspect of the service, or by affecting each other. Resolving such interferences automatically is a non-trivial task, especially since the formal definition of policies in TOSCA does not cover their specific effect. Therefore, we perceive resolving policy interferences a duty of the Cloud security officer who implements the policies in the Service Template. She has to ensure that dependencies and interferences are taken into account by providing policy implementations that are aware of these issues and behave accordingly.

Global and Local Policies: As described in our taxonomy in Sect. 4, a policy can be annotated at different layers in the service topology. It follows that a policy can be effective for multiple nodes or relationships at the same time. Therefore, these entities have to be made policy-aware by the Cloud service security officer even if they are not directly annotated with a policy themselves.

5.1 Cloud Service Lifecycle

In the TOSCA primer [15] and the CloudCycle project [12] the Cloud service lifecycle is described as shown in Fig. 4. The steps of this lifecycle are the building

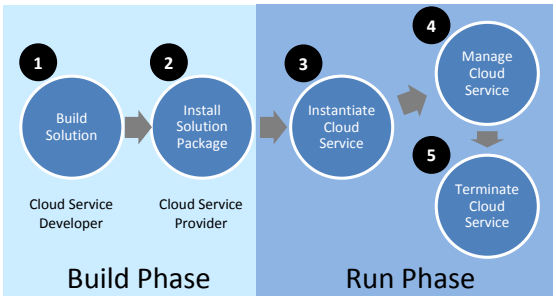

Fig. 4. Lifecycle of a cloud service, based on [12, 15]

of a solution package that represents the Cloud service (step 1). This Solution Package is installed in step 2. Afterwards the service can be instantiated multiple times (step 3) by calling the Build Plan and is managed by management plans later on (step 4). When the Cloud service is not needed anymore it is terminated by a Termination Plan (step 5).

We enhanced the Cloud service lifecycle with new steps for enabling secure Cloud Computing based on policies and offerings (see Fig. 5).

In the secure lifecycle a Cloud service developer bundles an existing service into a solution package, i.e., a CSAR file (step 1) which is ready to be provisioned in a Cloud environment. However, it is not yet secured. Therefore, a Cloud Service Security Officer secures the application by annotating policies and implementing their functionality through modified Plans or Implementation Artifacts. She combines different selections of policies and their configuration into Offerings and adds them to the policy annotation (step 2). A Cloud Service Provider installs this solution package in his TOSCA runtime and presents the included offerings to potential customers (step 3). This concludes the Build Phase of the Cloud Service.

In step 4 the Cloud Service Customer selects a solution package and chooses the desired offering (step 5) that determines the policies, which he requires. After this Selection Phase, the Run Phase of the Cloud Service begins. Each

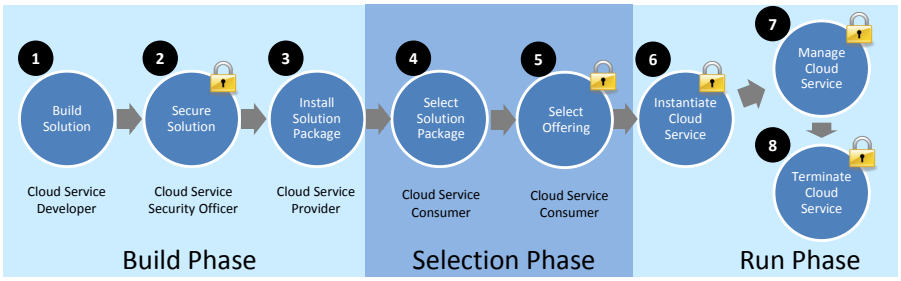

Fig. 5. Security enhanced lifecycle of a cloud service

time a new instance of the Cloud Service is requested, a Cloud Service Instance is instantiated (step 6) and managed (step 7) by the Cloud Service Provider. When the Cloud Service is no longer needed, the Customer issues its termination (step 8).

5.2 Method for Policy-Aware Cloud Service Provisioning

In this paper, we consider two approaches for implementing policies in an existing Service Template, which can be used in combination or oy themselves. They are performed by the Cloud Service Security Officer.

In the *Plan-based approach (P-Approach)* the Build, Management and Termination Plans are modified to execute additional operations, which implement the security features required by the annotated policies. The *Implementation Artifact based approach (IA-Approach)* on the other hand does not modify the plans, but rather replaces the Implementation Artifacts with security-enabled ones. These new Implementation Artifacts provide the same API as the previous ones, but the operations they provide perform additional steps that implement the required security features. Finally, the service topology is annotated with policies, which can be fulfilled by the replaced Implementation Artifacts and/or Plans. The concrete steps for the IA-Approach are described in Sect. 5.2 and for the P-Approach in Sect. 5.2.

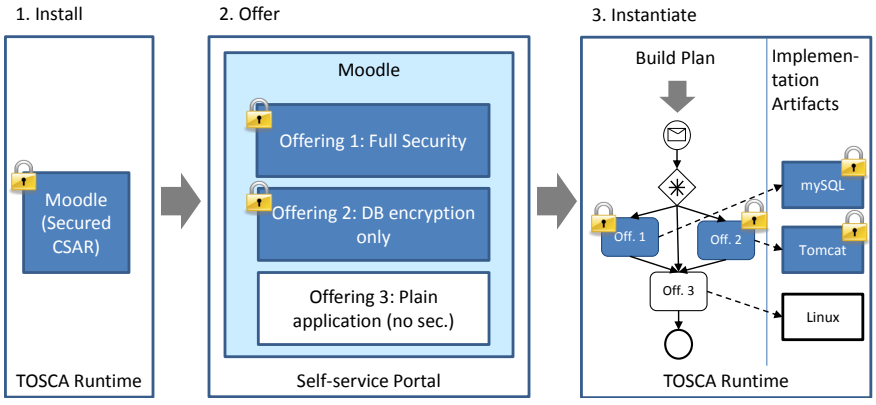

Fig. 6. Automated Policy Processing with Plans and Implementation Artifacts

Figure 6 shows the process of a policy-based Cloud Service instantiation. First, the Cloud service archive is installed in the TOSCA runtime, in our example, the Moodle application ("1. Install"). This application archive supports different offerings that determine which policies should be active and how they are configured. In our example, these are the Region Policy and the Encrypted Database Policy. The installed archive is then presented in the self-service portal

where the users can select their applications. Here different offerings can be selected for the Moodle application ("2. Offer"). Figure 6 shows three example offerings: Offering 1 provides Moodle with all security policies, offering 2 only provides database encryption and offering 3 would be the cheapest, because no security is requested.

Based on the selected offering the Build Plan receives different input parameters for the application instantiation ("3. Instantiate"). Using these parameters the selected offering is passed to the Build Plan as well as additional properties required for service instantiation.

Then the Build Plan is executed and checks with an `if`-statement whether a policy has to be enforced or not and processes the whole topology. Depending on the selected offering, the concrete Implementation Artifacts will be used. In Fig. 6 this is depicted by the arrows connecting the Build Plan activities and the called Implementation Artifacts.

5.2.1 Implementation Artifact Based Policy Enforcement: IA-Approach

In TOSCA, Implementation Artifacts realize service management operations provided by Node Types. When introducing policies, the execution of those management operations will be influenced. For example, depending on whether a database is required to be encrypted or not, its deployment and configuration has to be done differently. Therefore, we propose to support policies in TOSCA through policy-aware Implementation Artifacts.

To realize our approach, existing Implementation Artifacts have to be extended with additional policy enforcing implementations. The existing implementations will remain, as we still need the capability to provide the basic management operations. The extended Implementation Artifacts are thus comprised of two alternative implementations for each service management operation: one that is policy enforcing and one that is not. For each call of a service management operation the Implementation Artifact has to check if any policy has to be enforced. Depending on this check, the Implementation Artifact selects the appropriate implementation to execute the called operation. To realize this concept we propose a two-stage procedure which is given in Fig. 7.

First, a policy support check is performed during the CSAR deployment (steps 1 and 2). For this first stage, we extend the TOSCA processing flow of the container. When parsing a newly deployed TOSCA topology, the container passes information about Node Templates and their associated policies to the corresponding Implementation Artifacts. Each Implementation Artifact then responds if it is capable of enforcing the policies associated with the respective Node Template (step 3).

The second stage is executed by the Implementation Artifacts for each call of the service management operations which is represented by step 4 in Fig. 7. Implementation Artifacts are defined per Node Type; they are not bound to specific Node Templates. Therefore, the Implementation Artifact asks the container for the Node Template the current operation call is associated with, and if there are

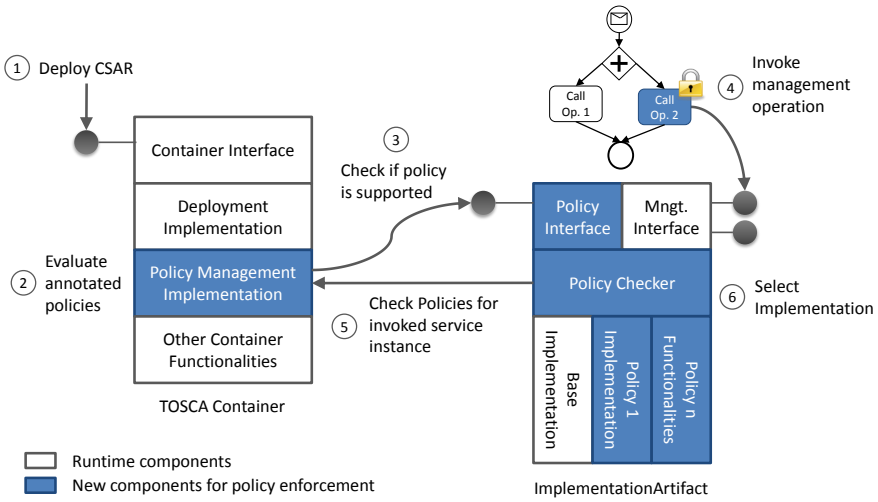

Fig. 7. Implementation Artifact Based Policy Enforcement

any policies registered for this Node Template under the current Offering (step 5). Depending on the result, the corresponding implementation for the called service management operation is selected and executed (step 6).

5.2.2 Plan-Based Policy Enforcement: P-Approach

A policy-aware Management Plan is an imperative way to enforce policies. For each policy, the Plan triggers the appropriate steps. These steps are based on the semantics of each policy. Therefore, there is no general rule for writing enforcement instructions. For instance, when enforcing the Region Policy the Plan will insert a request for the specified region into the message it sends to the Cloud Provider. In the case of the Encrypted Database Policy, the Plan will execute additional configuration steps for enabling encryption when setting up the database.

The different steps of P-Approach are shown in Fig. 8. Step 1 is the deployment of a CSAR to the TOSCA Container using the container interface. Then in step 2 the policies annotated in the topology are evaluated and are stored and made available by the Policy Management Component. Step 3 is the presentation and offering of the installed service with all its different offerings in the self-service Portal - a web-portal for the customer to select the desired service and offering for the individual security requirements. After the customer selected the service with a concrete offering a message is generated and sent to the Build Plan (input message). This message starts a new workflow instance that processes the build plan (step 4).

The Build Plan then determines which policies should be enforced, it reads the selected Offering from its input message and checks the Policy Management

Implementation to determine which policies this Offering includes and what their specific configuration will be (step 5). Then, the Build Plan is executed and based on the activated policies different branches in the Build Plan are executed (step 6).

For the Plan to determine which policies should be enforced, it reads the selected Offering from its input message and checks the Policy Management Implementation to determine which policies this Offering includes and what their specific configuration has to be.

The Cloud security officer has to ensure that the plan conforms to the security annotations given in the topology. One way to check is compliance checking of processes as presented by Schleicher et al. [18].

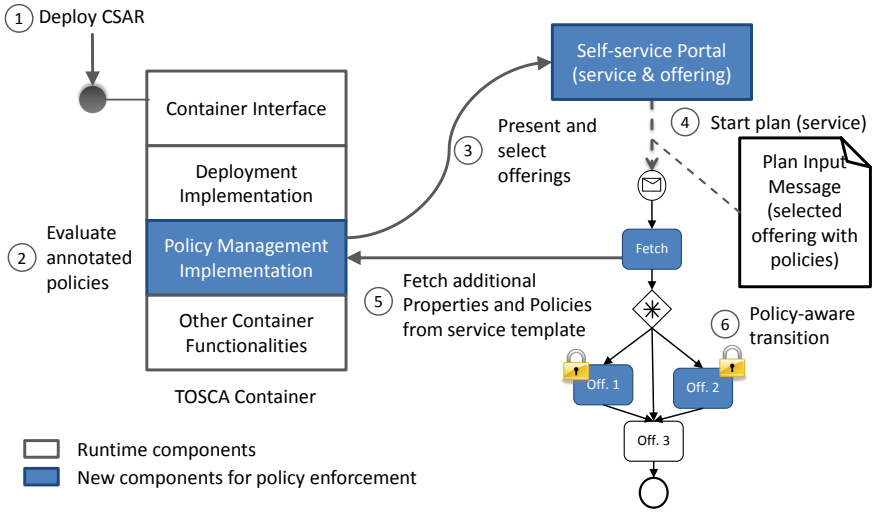

Fig. 8. Plan Based Policy Enforcement

5.3 Discussion

In the previous sections, two different approaches were presented to implement policies. These approaches allow the enforcement of policies in a TOSCA environment. However, it should by carefully considered which approach is used for realizing which policy.

Regarding reusability TOSCA allows NodeTemplates or NodeTypes to be reused in different topologies. Similarly, policies implemented with the IA-Approach can be reused very easily as they are implemented locally on the IA level. However, Build Plans are designed globally, i.e. with the whole topology in mind. Hence, the P-Approach is service specific and can not be reused for different service topology.

On the other hand, in more complex scenarios, it might be necessary to monitor the whole service on a global level in order to determine if enforcement actions have to be triggered. However, usually IAs are only aware of the local state they are in. This limits their capability to make global decisions concerning policy enforcement that depend on information from other IAs of the topology. This global control can be achieved with the P-Approach. The management plans can be designed to gather the information for policy enforcement from all required components. Based on this information the plans can decide when enforcement actions have to be performed. So the P-Approach is on an other abstraction level and has global knowledge, whereas the IA-Approach works on a detailed level considering propertied of one concrete component.

6 Conclusion

In this paper we first introduced TOSCA a new standard by OASIS for defining Cloud services. In order to allow the definition of security policies for Cloud services, we introduce in this paper a formal policy definition based on a taxonomy defining the stage, layer, and effect of policies. Multiple policies are combined into an offering together with a formal TOSCA Cloud service definition. The customer can now chose such an offering that fits his requirements. We then present a method for policy aware Cloud service provisioning consisting of a security-enhanced Cloud service lifecycle and two approaches (P-Approach and IA-Approach) for processing policies during the service deployment in a TOSCA runtime. We concluded by giving a discussion about the differences between the two approaches.

Acknowledgements

This work was partially funded by the BMWi Trusted Cloud project CloudCycle (01MD11023) and the BMWi IT2Green project Migrate! (01ME11055).

References

1. Beisiegel, M., Booz, D., Colyer, A., Hildebrand, H., Marino, J., Tam, K.: SCA – service component architecture (March 2007)
2. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing 16(03), 80–85 (May 2012)
3. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: A visual notation for application topologies based on TOSCA. In: Cooperative Information Systems. Springer (2012)
4. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wieland, M.: Policy-aware provisioning of cloud applications. In: Conference on Emerging Security Information, Systems and Technologies. IARIA (2013)
5. Garbani, J., Mendel, T., Radcliffe, E.: The writing on IT's complexity wall (2010), Forrester Research

6. Garlan, D., Monroe, R., Wile, D.: Acme: an architecture description interchange language. In: Conference of the Centre for Advanced Studies on Collaborative Research. IBM Press (1997)
7. Leymann, F.: Cloud computing. it – Information Technology 53(4) (2011)
8. Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S.: Moving applications to the cloud: an approach based on application model enrichment. Int. J. Cooperative Inf. Syst. 20(3), 307–356 (2011)
9. Machiraju, V., Dekhil, M., Wurster, K., Garg, P.K., Griss, M.L., Holland, J.: Towards generic application auto-discovery. In: Hong, J.W.K., Weihmayer, R. (eds.) Network Operations and Management Symposium. IEEE (2000)
10. Mell, P., Grance, T.: The NIST definition of cloud computing. Recommendations of the National Institute of Standards and Technology Special Publication 800-145, 7 (2011)
11. Mietzner, R.: A method and implementation to define and provision variable composite applications, and its usage in cloud computing. Ph.D. thesis, Universität Stuttgart (2010)
12. Niehues, P., Kunz, T., Posiadlo, L.: Das CloudCycle-Ökosystem. Tech. rep., CloudCycle (2013)
13. Nowak, A., Binz, T., Fehling, C., Kopp, O., Leymann, F., Wagner, S.: Pattern-driven green adaptation of process-based applications and their runtime infrastructure. Computing pp. 463–487 (Feb 2012)
14. OASIS: OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0 Committee Specification 02 (2013), `http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs02/TOSCA-v1.0-cs02.html`
15. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0 (January 2013), `http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html`
16. Object Management Group: Unified modeling language 2.1.2 super-structure specification. Specification Version 2.1.2, Object Management Group (November 2007)
17. Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do internet services fail, and what can be done about it? In: USENIX Symposium on Internet Technologies and Systems (2003)
18. Schleicher, D., Leymann, F., Schneider, P., Schumm, D., Wolf, T.: An Approach to Combine Data-Related and Control-Flow-Related Compliance Rules. In: Conference on Service Oriented Computing & Applications. IEEE (Dec 2011)
19. Sunyaev, A., Schneider, S.: Cloud services certification. Commun. ACM 56(2), 33–36 (Feb 2013)
20. Takabi, H., Joshi, J., Ahn, G.J.: Securecloud: Towards a comprehensive security framework for cloud computing environments. In: Computer Software and Applications Conference Workshops (2010)
21. Unger, T., Mietzner, R., Leymann, F.: Customer-defined service level agreements for composite applications. Enterp. Inf. Syst. 3(3), 369–391 (Aug 2009)
22. Waizenegger, T., Wieland, M., Breitenbücher, U.: Towards a policy-framework for provisioning and management of cloud services. In: Conference on Emerging Security Information, Systems and Technologies. IARIA (2013)

All links were last followed on 2013-05-29.