Deployment Enforcement Rules for TOSCA-based Applications

Michael Zimmermann, Uwe Breitenbücher, Christoph Krieger, and Frank Leymann

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

***Abstract*—In the context of Industry 4.0, gathering sensor data and using data analysis software can lead to actionable insights, for example, enabling predictive maintenance. Since developing these data analysis software requires some special expert knowledge, often external data scientist are charged for that. However, often the data to be analyzed is of vital importance and thus, must not leave the company. Therefore, applications developed and modeled as deployment models by third-parties have to be enforced to be executed in the local company's network. However, manually adapting a lot of these deployment models in order to meet the company's requirements is cumbersome, time consuming and error-prone. Furthermore, some kind of enforcement mechanism is required to really ensure that these data security and privacy requirements are fulfilled. Thus, in this paper, we present an approach considering these issues during the deployment time of the application. The presented approach is based on the Topology and Orchestration Specification for Cloud Applications (TOSCA), an OASIS standard enabling the description of cloud applications as well as their deployment. The approach enables the specification as well as the enforcement of reoccurring and generic requirements and restrictions of TOSCA-based declarative deployment models, without the need to adapt or modify these deployment models. The practical feasibility of the presented approach is validated by extending our open-source prototype OpenTOSCA, which provides a modeling tool, a TOSCA Runtime, as well as a self-service portal for TOSCA.**

***Keywords*–Cloud Computing; Application Provisioning; Automation; TOSCA; Security.**

I. Introduction

In the area of Internet of Things [1] and Industry 4.0 [2], the gathering of sensor data can lead to actionable insights by utilizing data analysis software, for instance, enabling predictive maintenance of cyber-physical manufacturing systems. However, the development of such analysis software for analyzing the gathered data requires special expert knowledge, for example, about implementing machine learning algorithms [3]. But, since companies often do not have this kind of knowledge and expertise for implementing such complex and domain-specific analysis software by themselves, they typically charge external data scientists to build the required software for them. Unfortunately, because of data security and privacy reasons as well as different company requirements and policies, often the gathered data to be analyzed is of vital importance for the company and must not leave the company and thus, can not be provided to third-parties, as for example, the data scientists [4]. Therefore, data scientists have to provide their developed software in a way, that enables the companies to automatically install and configure the analysis software, required middleware, and dependencies as well as to execute and link the software with the sensor data in their local company's infrastructure [5].

However, the data security and privacy requirements and policies as well as infrastructure information can differ from company to company or might be kept secretly as well. Therefore, third-party companies and data scientists can not always take these requirements and policies into account when developing the analysis software and creating the deployment models enabling the automated provisioning. Thus, the deployment models need to provide some configuration capabilities in order to be easily adaptable to the local infrastructure and environment of the respective company. Furthermore, with modern applications consisting of complex and heterogeneous components, it can become difficult to comply security requirements, especially when different deployment technologies are used [6] [7]. However, the enforcement of the defined security requirements needs to be ensured under all circumstances in order to secure the data. Regardless of whether the deployment model is created by a third-party company, an external data scientist, or even internally. Therefore, some possibility to easily specify such reoccurring requirements reflecting the company's policies as well as an automated enforcement mechanism are required. However, in a way that separates the modeling of requirements from the modeling of deployment models, since this again is a complex task and requires expert knowledge.

In this paper, we tackle the aforementioned issues. We present our concept of *Deployment Enforcement Rules* in order to specify reusable requirements and restrictions for TOSCA-based declarative deployment models. Furthermore, our approach ensures the enforcement of these requirements and restrictions during the provisioning of an application. Our approach is based on the Topology and Orchestration Specification for Cloud Applications (TOSCA), an OASIS standard enabling the description of cloud applications as well as their deployment [8]. By extending an existing deployment technology, our approach enables the fully automated deployment of cloud and IoT applications, while enforcing security requirements. Our approach is validated by a prototypical implementation based on the OpenTOSCA Ecosystem [9] [10].

The remainder of this paper is structured as follows:
In Section II, the fundamental concepts of the standard TOSCA are explained. TOSCA is used within our approach as a cloud and IoT application modeling language. Afterward, in Section III our approach is motivated by illustrating a TOSCA-based Industry 4.0 scenario. In Section IV, our approach of Deployment Enforcement Rules for declarative deployment models based on TOSCA are explained. In Section V, our approach is validated by presenting a prototypical implementation based on the OpenTOSCA Ecosystem. In Section VI, an overview of related work is given. Finally, Section VII concludes this paper and presents an outlook on future work.

II. Topology and Orchestration Specification for Cloud Applications

Since our work is based on TOSCA, in this section, the OASIS standard TOSCA is explained. The TOSCA standard enables the automated deployment, as well as management of cloud and IoT applications. In this section, we only briefly describe the fundamental concepts of TOSCA required to understand our presented approach. A detailed overview of TOSCA can be found in the TOSCA Specifications [8] [11], the TOSCA Primer [12] and an overview by Binz et al. [13].

A. Nodes, Relationships, Types, and Templates

Using TOSCA, the components of an application – software components as well as infrastructure components – and their relationships to each other can be described in a standardized and portable manner. The modeled structure of an application is defined by so-called *Topology Templates*. A Topology Template is a directed graph and consists of nodes and directed edges. The nodes represent the components of the application and are called *Node Templates*. A Node Template could be, for example, an Apache Tomcat, an Ubuntu virtual machine, or an OpenStack hypervisor. The Node Templates are connected by the edges, which are called *Relationship Templates* and specify the relations between the Node Templates. A Relationship Template could define, for example, a “hostedOn”, “dependsOn”, or “connectsTo” relation between two Node Templates. Thus, Relationship Templates are specifying the structure of an application. In order to enable reusability, the semantics of Node Templates and Relationship Templates are defined by *Node Types* and *Relationship Types*. Node Types as well as Relationship Types are reusable entities allowing to define *Properties*, as well as *Management Operations*. A NodeType “OpenStack”, for example, may have defined Properties for specifying the URL required for accessing a running OpenStack instance as well as credential information, such as a username or a password. The Management Operations defined by a Node Type can be bundled in interfaces and can be invoked in order to manage the instances of this component. For example, an “Apache Tomcat” Node Type may define a Management Operation “install” in order to install the component itself as well as a Management Operation “deployApplication” in order to deploy an application on it. Furthermore, a cloud provider or hypervisor Node Type typically provides Management Operations in order to create virtual machines (“createVM”) as well as to terminate virtual machines (“terminateVM”).

B. Implementation Artifacts and Deployment Artifacts

Two kinds of artifacts are defined by TOSCA: (i) *Implementation Artifacts (IAs)*, as well as (ii) *Deployment Artifacts (DAs)*. The Management Operations defined by Node Types are implemented by IAs. An IA itself can be implemented using various technologies, for instance, as a Web Services Description Language (WSDL)-based web service, a shell script, or by using configuration management technologies, such as Ansible [14] or Chef [15]. Generally, three kinds of IAs can be distinguished, dependent on the way they are processed: (i) IAs, that are copied to the target environment of the application and are executed there, for example, shell scripts. (ii) IAs, that are deployed and also executed in the *TOSCA Runtime* environment (cf. Section II-E), for example, SOAP-based web services. These IAs typically use remote access protocols, for instance SSH or SFTP in order to manipulate components, perform operations on it, and to transfer files on a virtual machine for example. (iii) IAs, that are just referred within a Topology Template, since the modeled component is already running somewhere. Such IAs are, for example, a web service API of a cloud provider or a hypervisor, such as OpenStack.

The TOSCA standard also defines so-called Deployment Artifacts. In contrast to IAs, DAs implement the business functionality of a Node Template. For example, the DA of a PHP application node could be a *.ZIP file, which contains the PHP files, images, and all other files required for provisioning the PHP application. Another example of a DA would be a *.WAR file, implementing the java web application of a node. Deployment Artifacts are typed and may define additional information, such as the location of the corresponding binary.

C. Management Plans

In order to create or terminate an instance of a modeled TOSCA-based application or to automate the management, so-called *Management Plans* are used. A Management Plan defines all tasks as well as the order in which these tasks need to be executed in order to fulfill a specific management functionality, for example, to provision a new instance of the modeled application. Therefore, the Management Operations which are specified by Node Types and are implemented by the corresponding Implementation Artifacts are invoked by Management Plans. The TOSCA standard allows to use any arbitrary process modeling language, but recommends to use workflow languages such as the *Business Process Execution Language (BPEL)* [16] or the *Business Process Model and Notation (BPMN)* [17]. There is also a BPMN extension called *BPMN4TOSCA* [18], [19], which is explicitly tailored for describing TOSCA-based deployment and management plans.

D. Cloud Service Archives

The TOSCA specification also defines a portable as well as self-contained packaging format, so-called *Cloud Service Archive (CSAR)*. A CSAR enables to package all aforementioned artifacts, templates, type definitions, plans, and all other additionally required files together into one archive, which technically is a .zip file. Therefore, a CSAR contains everything required for enabling the automated provisioning and management of the modeled application. Moreover, because of the mentioned characteristics, CSARs also enable to easily share and distribute such modeled TOSCA-based applications, for example, between colleagues, project partners, or to customers.

E. TOSCA Runtimes

The processing and execution of CSARs is done by standard-compliant TOSCA Runtimes. However, there are two different approaches for provisioning a TOSCA-based application: (i) declaratively as well as (ii) imperatively [20]. Therefore, there are also two types of TOSCA Runtimes. A TOSCA Runtime can either process a Topology Template (i) declaratively by interpreting and deriving the actions required to provision the modeled application directly from the Topology Template itself. In this case, no Management Plan is required. Furthermore, a TOSCA Runtime can also process a TOSCA-modeled application (ii) imperatively by using Management Plans associated with a Topology Template, specifying which Management Operations need to be executed in which order.

III. Motivating Scenario

In this section, a TOSCA-based motivation scenario is described. This motivation scenario is used throughout the entire paper for explaining and demonstrating our approach. Figure 1 illustrates the motivation scenario as a TOSCA Topology Template. The modeled application abstractly depicts an exemplary Industry 4.0 scenario with a data analytics stack on the left side (*PredictionService*) and the data to be analyzed on the right side (*MySQLDB*) of the illustrated topology.

In Industry 4.0, for example, manufacturing data gathered during the production process can be analyzed in order to enable predictive maintenance of cyber-physical manufacturing systems. The analytics stack in Figure 1 consists of an *Apache Flink* Node Template, which is hosted on (specified by using a "hostedOn" Relationship Template) an *Ubuntu* virtual machine Node Template. The virtual machine is managed by the hypervisor OpenStack, which should be operated locally in the infrastructure of the company. In general, Apache Flink is an analytics platform with batch as well as stream processing capabilities enabling the integration, processing, and analyzing of data sources, such as MySQL databases. In our motivation scenario, the MySQL database used to store the generated analysis data is also running on an Ubuntu virtual machine, which is hosted on the same OpenStack instance as the Prediction Service. Since, the Prediction Service needs to establish a connection to the MySQL database in order to access and analyze the data, both Node Templates are connected using a "connectsTo" Relationship Template. Furthermore, required credentials, for instance, the username ("DBMSUsername") or password ("DBMSPassword") of the database are provided as Properties. In oder to instantiate an Ubuntu virtual machine, the OpenStack Node Template exposes Management Operations, like for example "createVM". Management Operations can use Properties, predefined during the modeling time, as input in order to customize the specification of a component, for example, the amount of RAM or hard disk capacity in case of a virtual machine. But also during the provisioning time, the modeled application can still be customized. This can be achieved by setting the value of any arbitrary Property to "getInput()". The values of such defined Properties are requested when the provisioning is instantiated. The advantage of this is that a parameterizable CSAR containing the Topology Template and all other required files can be distributed among business partner or customers. In Figure 1, for example, the username ("HUsername"), the password ("HPassword"), as well as the endpoint ("Endpoint") are defined as "getInput()" in order to enable the adaption of these Properties according to the company's respective infrastructure. Due to the fact, that the data to be analyzed can contain business-critical information that has to be protected and must not leave the company, the OpenStack needs to be operated within the local environment of the company. Therefore, in our scenario the credentials and the endpoint of the local OpenStack are not predefined and need to be provided during provisioning time of the application.

However, the enforcement of the local deployment needs to be ensured under all circumstances in order to secure the data. Regardless of whether the Properties are provided manually when the provisioning of the application is instantiated or are already predefined in the deployment model. Therefore, some possibility to specify such requirements and restrictions regarding the deployment model as well as an enforcement mechanism are required to achieve that. Of course, besides the requirement to restrict the physical location of the provisioning of the application, other requirements are imaginable as well. For example, a requirement specifying that some components are only allowed to be hosted on specific operating systems, because they might provide some special security features. Using TOSCA, it is possible to specify such non-functional requirements, for example, by defining corresponding *Policy Types* and *Policy Templates*. However, they need to be modeled directly within the Topology Template and are attached to Node Templates for which the policy needs to be fulfilled. Therefore, in order to meet the respective requirements, for every CSAR the Topology Model respectively the TOSCA definition files must be adapted according to the company's business requirements and policies. However, manually adapting a lot of CSARs in order to meet the same requirements is cumbersome, time consuming and error-prone. Therefore, an alternative option enabling the easily specification as well as enforcement of these reoccurring and generic requirements is required. The defined requirements should be appendable to a CSAR without adapting TOSCA definition files, but just by adding additional files defining the requirements. Also, besides *Whitelisting Rules*, defining what is allowed, also *Blacklisting Rules*, defining what is forbidden should be supported. In the following section we explain our idea of generic and reusable *Deployment Enforcement Rules* tackling these issues in detail.

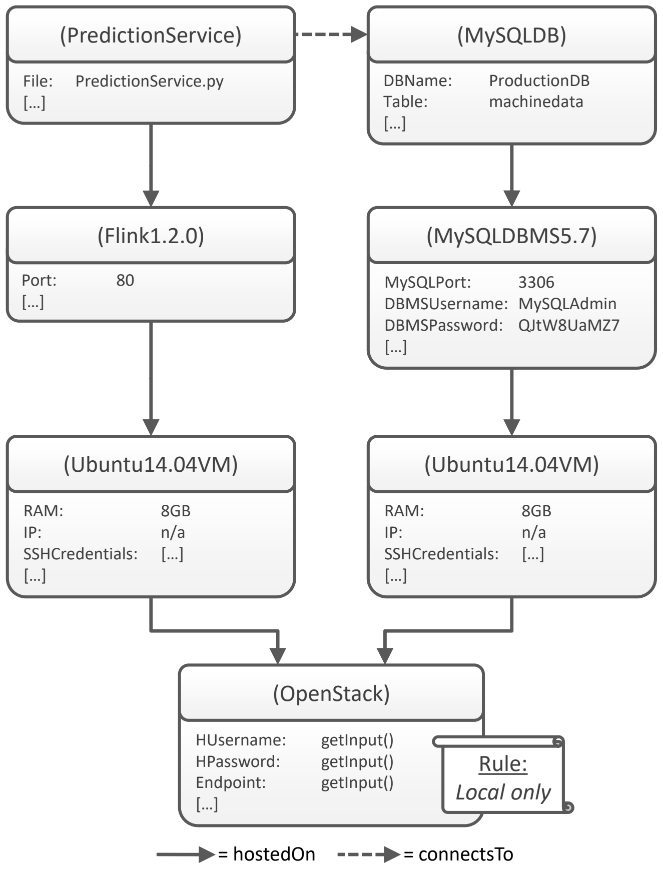

Figure 1. Analytics functionality as well as the database containing the dataset to be analyzed should be hosted on the local infrastructure of the company due to data security and privacy requirements.

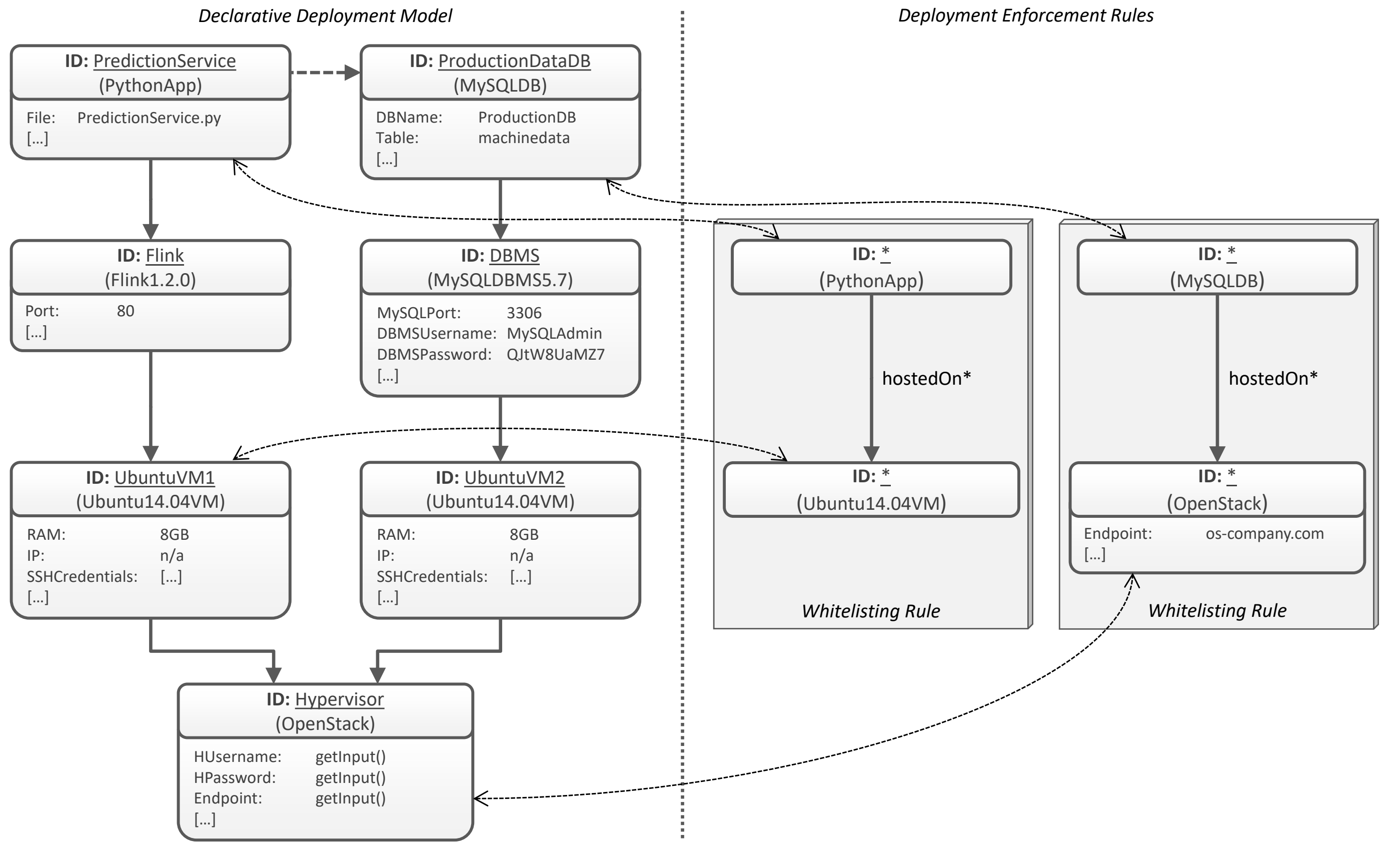

Figure 2. Concept of Deployment Enforcement Rules for defining requirements for declarative deployment models that have to be fulfilled to deployment time.

IV. Deployment Enforcement Rules

In this section, our approach of *Deployment Enforcement Rules* for specifying requirements regarding the deployment model are explained. First, an overall presentation of the Deployment Enforcement Rules concept is given, following the TOSCA-based motivation scenario described in the previous section. After that, the full potential of the approach is shown by combining *Whitelisting Rules* together with *Blacklisting Rules* in order to define more complex requirements and restrictions.

The main goal of our Deployment Enforcement Rules approach is to enable the creation of generic and reusable rules for automatically ensuring the fulfillment of specified requirements and restrictions regarding the deployment model of an application. For example, requirements restricting the physical location where an application is allowed to be provisioned or requirements restricting that just specific operating systems are allowed to be used or are forbidden. Furthermore, the Deployment Enforcement Rules should be specified separately from the deployment models in order to be easily appendable to the existing deployment model, but without the need to adapt or modify the respective deployment models. Thus, no expertise about the deployment model, the contained components, or the used deployment technologies are required in order to make the deployment models compliant to the company's security policies. Only the requirements and restrictions that should be taken into account when provisioning the modeled application must be known for defining the Deployment Enforcement Rules. Once defined, these rules can be reused over and over again.

A. *Overview of the Approach*

The concept of our approach is illustrated in Figure 2, following the motivation scenario introduced in Section III. On the left side of the figure, the declarative deployment model for provisioning the analysis software as well as the database containing the data to be analyzed is shown. The deployment model is the same as already described in the previous section, however now also providing the IDs of the components, such as “Hypervisor” or “UbuntuVM1”. The Node Types are defined within the brackets, e.g., “OpenStack” or “Ubuntu14.04VM”. On the right side of the figure, two exemplary Deployment Enforcement Rules are illustrated. Since both rules explicitly are defining what is allowed instead of what is forbidden, both shown rules are Whitelisting Rules. In the shown example, the left rule defines, that a component of the type “PythonApp” is only allowed to be installed on a virtual machine running Ubuntu 14.04., because this might be the stablest and securest Ubuntu version available. The rule on the right side defines, that a MySQL database must be hosted on an OpenStack instance running on the specified “Endpoint” *os-company.com*, because this is the endpoint where the company's local OpenStack instance is running. Therefore, the database containing the data can only be hosted within the company's local infrastructure and thus, the data is not leaving the company's sovereignty. In both rules, the ID of the components is not defined, which means that in these cases only the Node Types are taken into account for deciding if the rules are fulfilled or not, independently of the ID of the specific Node Template in the deployment model.

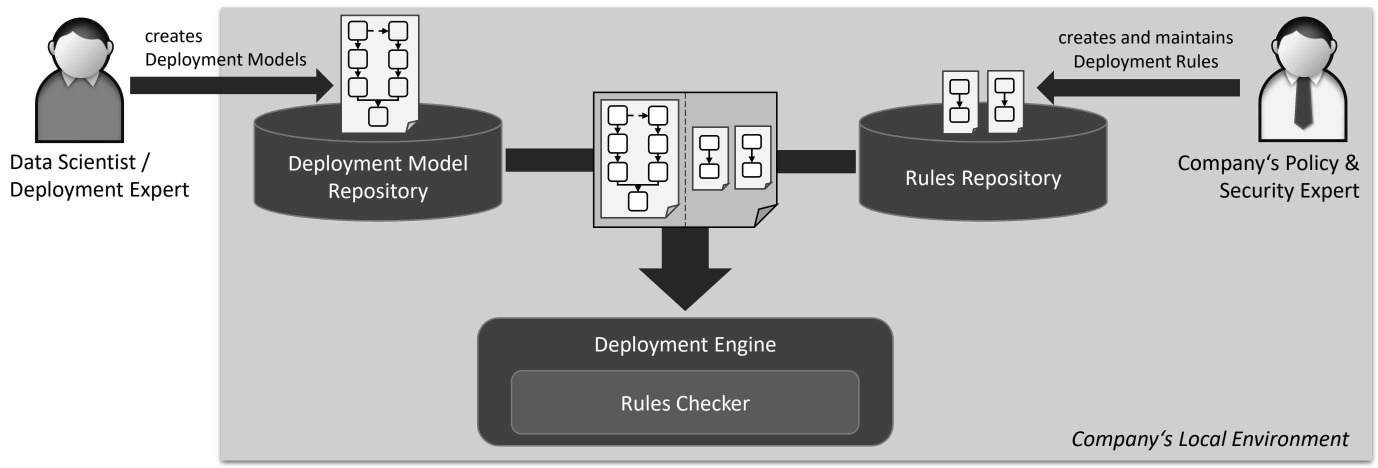

Figure 3. Overview of the Deployment Enforcement Rules approach, showing involved roles, models, and components.

Both Deployment Enforcement Rules shown in Figure 2 are defined using a transitive relation ("hostedOn*"). Since the middleware, dependencies, and other required components are not necessarily important for the fulfillment of security requirements, using the transitive relation enables to only specify the relevant components in order to define the Deployment Rules. Regarding the deployment model shown in Figure 2, after matching the "MySQLDB" node in the Deployment Enforcement Rule with the "ProductionDataDB" in the deployment model, the "hostedOn" relations in the deployment model are traced downwards the modeled stack until the "OpenStack" node is found – or no further "hostedOn" relation can be found. When the "OpenStack" node is found, it is checked whether the value of the "Endpoint" property defined in the Deployment Enforcement Rule is matching the actual value of the "Endpoint" property in the deployment model or not. Since properties can already be predefined in the deployment model (cf. "MySQLPort" in node "DBMS" of Figure 2) or are only provided when the provisioning is instantiated (cf. "Endpoint" in node "Hypervisor" of Figure 2), the rules need to be checked for fulfilling to deployment time, thus, they are called Deployment Enforcement Rules. To sum up, the use of transitive relations enable to specify only the components relevant for a specific Deployment Enforcement Rule and therefore, ease the creation of Deployment Enforcement Rules as well as increase the reusability of already existing Deployment Enforcement Rules.

The involved roles, models, and components of the approach are shown in Figure 3. On the left side, a possibly external data scientist or deployment expert is shown. This person is responsible for implementing the application and creating the deployment model. Possessed deployment models can be stored within the company's local environment using the *Deployment Model Repository*. On the right side a company's internal policy and security expert is shown, which is responsible for creating and maintaining the Deployment Enforcement Rules according to the company's polices and restrictions. Again, the created rules can be persistently stored in a local *Rules Repository*. Deployment models and Deployment Enforcement Rules are combined beforehand the deployment in order to ensure the enforcement of the security policies of the company. Therefore, the *Deployment Engine* contains a *Rules Checker* for checking if the specified Deployment Enforcement Rules are fulfilled.

B. *Further Examples, Blacklisting Rules, and Inheritance*

In this subsection two more exemplary Deployment Enforcement Rules are presented. While on the left side of Figure 4 another Whitelisting Rule is illustrated, on the right side a Blacklisting Rule is shown. Furthermore, the support of inheritance for Deployment Enforcement Rules is demonstrated.

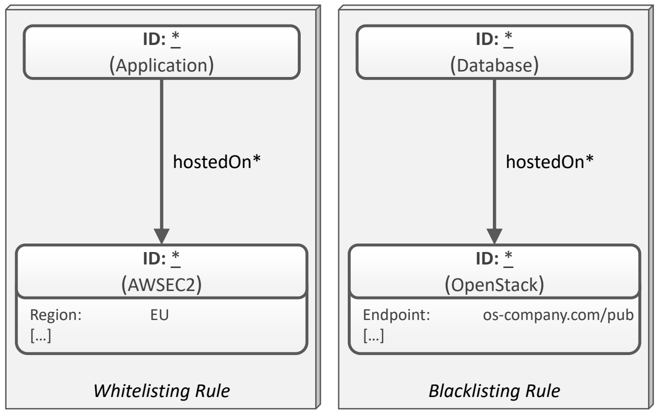

Figure 4. Exemplary Whitelisting Rule and Blacklisting Rule.

The Whitelisting Rule restricts the deployment of applications in a way that they are only allowed to be hosted on an AWS EC2 instance operated in the EU region. The rule also shows the usage of inheritance in order to create generic and reusable rules. Here, the "Application" Node Type is used, which can be seen as a super type for any other application component, such as the "PythonApp" from Figure 2. Thus, the approach enables to create very generic rules as well as highly unique rules, for example, by defining the specific Node Type as well as providing the ID of the component to be checked. The Blacklisting Rule on the right side forbids that any database is hosted on the OpenStack instance running on the "Endpoint" *os-company.com/pub*, since this might be an OpenStack instance accessible from outside the company's infrastructure and thus, the data would not be secure there. As shown in this subsection, depending on the concrete requirement, our approach enables to define and use Whitelisting as well as Blacklisting Rules.

V. Validation & Prototype

In this section, we present our implemented prototype supporting the modeling and enforcing of Deployment Enforcement Rules. The prototype validates the practical feasibility of our proposed approach presented in the previous section. While in the first subsection, the general architecture as well as the components of the prototype are introduced, in the second subsection details of the concrete implementation are presented.

A. System Architecture

A conceptual architecture of our prototype is illustrated in Figure 5. The prototype consists of four main components: (i) the *modeling tool*, (ii) the *repository*, (iii) the *self-service portal*, and (iv) the *deployment engine*. By using the modeling tool, a user can graphically create and maintain deployment models as well as required reusable elements, such as relations and component types. Furthermore, the modeling tool also enables to define and maintain Deployment Enforcement Rules. The modeling tool is connected with the repository. In the repository, the created deployment models, relations, component types, as well as Deployment Enforcement Rules can be persistently stored. The self-service portal is used to chose an available deployment model of an application and to instantiate the deployment of it. Therefore, the self-service portal has access to the repository. Furthermore, not yet specified property values (cf. Section III) can be provided here. The deployment engine consumes deployment models in order to deploy the defined applications. Moreover, the deployment engine contains the *rules checker* component, which is responsible for checking whether the Deployment Enforcement Rules are fulfilled for the processed deployment models during the deployment time.

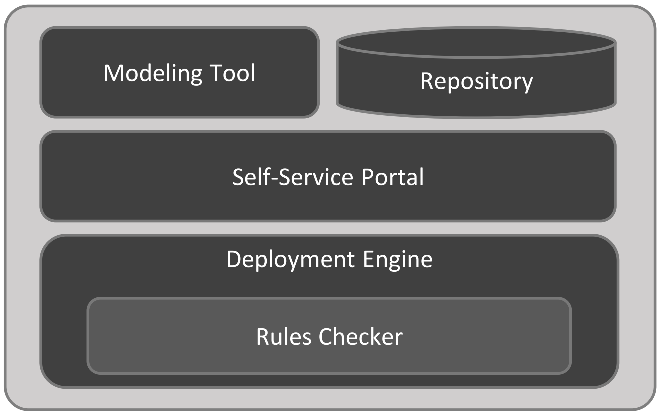

Figure 5. Architectural overview of the prototype.

B. Prototypical Implementation

Our prototype is based on the *OpenTOSCA Ecosystem* and extends the *OpenTOSCA Container* [9] component. OpenTOSCA is a standards-based TOSCA Runtime Environment, consisting of three main components: (i) *Winery* [10], (ii) *Vinothek* [21], and (iii) OpenTOSCA Container. Winery is a graphical tool for modeling and managing TOSCA Topology Templates as well as Node Types, Relationship Types and so on. Furthermore, Winery enables to package the topology as well as all required files into a CSAR and export it. Technically, Winery is implemented using Java 1.8 and is available as Web Application Archive (WAR). From an architectural perspective, Winery is split into two components: (i) *Topology Modeler*, the graphical front end for modeling the topologies and (ii) *Winery Repository*, which is the back end of Winery and enables the persistently storing of all files. Furthermore, since the same elements of the TOSCA standard are required for modeling Deployment Enforcement Rules as for modeling TOSCA Topology Templates, such as Node Template and Relationship Templates, Winery can also be used to model, store, as well as to export Deployment Enforcement Rules.

OpenTOSCA Container is the deployment engine of our prototype. It processes the exported CSARs from Winery, interprets the contained TOSCA deployment models, deploys Implementation Artifacts as well as Management Plans, and provisions the modeled application. In order to validate the practical feasibility of our proposed approach, we implemented the *Rules Checker* as an additional component of the OpenTOSCA Container. The Rules Checker component is responsible for checking if the Deployment Enforcement Rules presented in this paper are fulfilled or not. Therefore, the nodes, relations, properties, as well as the overall structure of the specified Deployment Enforcement Rules are checked against the Topology Template that should be provisioned. If the Deployment Enforcement Rules are fulfilled, the deployment of the modeled application can be continued. However, if the Deployment Enforcement Rules are unfulfilled, e.g., due to not matching endpoint properties in case of a Whitelisting Rule, the deployment is terminated and a corresponding error message is displayed. Afterwards, in case of not matching properties, these properties breaking the rules can be adapted in order to fulfill the rules and the deployment can be initiated again. Technically, the OpenTOSCA container as well as the Rules Checker component are implemented using Java 1.8 and are based on the OSGi Framework Equinox [22], a Java-based runtime environment enabling to build modular applications.

Vinothek is a self-service portal, providing a graphical user interface for enabling the end user to choose an available application and start the provisioning of it. If information are missing, such as required endpoint properties, a username, or a password, the user initiating the provisioning can insert this missing information here. Vinothek is also implemented using Java Server Pages (JSPs) and packaged as a WAR and thus, can be easily deployed on a web container such as Tomcat.

To sum up, we implemented our concepts within the OpenTOSCA Ecosystem, which already was able to process TOSCA Topology Templates and provision the modeled applications. In this work, we further extended the prototype by adding the additional Rules Checker component to also support the provisioning under consideration of security-related requirements and restrictions by supporting Deployment Enforcement Rules. All three mentioned OpenTOSCA components are open-source and can be obtained from GitHub (https://github.com/OpenTOSCA).

VI. Related Work

In this section, we present related work on our approach of enforcing company defined data security and privacy requirements during the deployment time of an application.

Walraven et al. [23] present PaaSHopper, which is a policy-driven middleware platform for developing and deploying multi-tenant SaaS applications in multi-PaaS environments. Based on the current context of stakeholder defined properties the middleware decides, on which parts in a multi-cloud a given request is processed or data will be stored. To achieve policy-awareness, PaasHopper middleware includes a policy-driven execution layer consisting of the two main components, the *Dispatcher* and the *Policy Engine*. Driven by the current context of defined policies, the *Dispatcher* selects an adequate component in a multi-cloud on which a request is processed or data will be stored. To do so, the dispatcher uses the *Policy Engine* to select a component instance that complies with the current context of policies. Contrary to our approach, modeling and enforcing data security and privacy requirements are restricted to applications that are deployed on PaaS solutions. Moreover, the restrictions that can be defined are limited to processing and storage of data, whereas our approach enables the specification of various requirements and restrictions.

Képes et al. [24] present an approach of enforcing specified non-functional security requirements during the provisioning phase of applications. For example, access restrictions or secure password requirements. These requirements are specified in form of policies that are attached to the Node Templates of a TOSCA Topology Template. Subsequent a Policy-Aware Provisioning Plan Generator transforms a given template into an executable policy-aware provisioning plan. In order to provide the required technical activities for the policy-aware provisioning, the Plan Generator provides a plugin system for implementing reusable policy aware-deployment logic. In difference to our loosely coupled deployment rules, their generated policy-aware provisioning plans are tightly coupled with the TOSCA Topology Templates they are generated from.

A similar approach to define non-functional security restrictions is presented by Blehm et al [25]. They also define security restrictions by means of policies attached to TOSCA Topology Templates. In addition they implemented policy specific services to ensure that the security restrictions are adhered. Again, the main difference to our approach is that the non-functional requirements are not separated from the deployment model but are directly attached to it. Thus, the deployment models need to be manually adapted to meet the company‘s requirements.

Waizenegger et al. [26] present the *IA-Approach* and the *P-Approach* to implement TOSCA-based security policy enforcement. The IA-Approach extends IAs by implementing the already existing Management Operations again with additional policy enforcing steps. The P-Approach extends the Plan required for provisioning the application with additional policy enforcing activities. Unlike to our approach, in their approach the policy enforcing elements are not separated from the deployment model but are directly attached to IAs or Plans.

Fischer et al. [27] present an approach to ensure compliance of application deployment models during their design time on the basis of the TOSCA standard. Similar to our approach, they aim to separate concerns about the knowledge base of a company's compliance requirements and the technical expertise of modeling applications, so that compliance experts can define compliance rules that can then be used to ensure compliance in deployment models. To achieve this, they introduce the concept of *Deployment Compliance Rules* which provide a means to ensure that deployment structures are conform with a company's compliance requirements. A deployment compliance rule consists of an Identifier graph to identify a compliance-relevant area in application deployment models and a Required Structure graph to define the allowed structure for the given compliance-relevant area. Unlike to our approach, the compliance checking of the deployment model is done during their design time. Moreover, Deployment Compliance Rules only allow to model allowed deployment structures and do not provide a means to define structures that are explicitly not allowed in a deployment model. Furthermore, the concept of transitive relations to ensure high reusability as well as faster modeling of the rules is not supported in their approach.

There are different approaches to specify and enforce certain requirements and restrictions in business process models. Fellmann and Zasada [28] conduct a literature review to provide an overview of the state-of-the-art approaches for mapping a company's compliance rules to business process models. Koetter et al. [29] present the concept of a Compliance Descriptor that links laws, regulations, and company intern restrictions to their technical implementation. Thereby, a Compliance Descriptor provides a means to consider the phases design-time, deployment and run-time of a business process life cycle. Depending on the phase different technologies are used for the technical implementation of the compliance requirements. Linear temporal logic (LTL) is used for design-time rules, TOSCA for requirements during the deployment phase and ProGoalML for run-time monitoring. Schleicher et al. [30] introduce the concept of *Compliance Domains* which can be used to model data restrictions for runtime infrastructures, such as different types of cloud environments or local data centers. In their approach, areas of business processes, modeled in Business Process Model and Notation (BPMN), can be marked by compliance experts with Compliance Domains that contain certain service level agreements and compliance rules that needs to be met. Based on this information, a graphical business process modeling tool can enforce the defined requirements during design time and notify the modeler if a selected runtime environment or data that enters a compliance domain violates them. For the most part, their work focuses on the restriction of data flows on the level of business process models, while our approach provides a method for enforcing security and privacy requirements on the level of declarative deployment models.

VII. Conclusion and Future Work

In this paper, we presented our approach of Deployment Enforcement Rules enabling the specification as well as the automated enforcement of reoccurring requirements and restrictions of declarative deployment models. For demonstrating the approach we used the OASIS standard Topology and Orchestration Specification for Cloud Applications (TOSCA). The approach allows to specify the Deployment Enforcement Rules separately from the deployment models and without the need to adapt or modify any deployment model at all. We showed, that by using transitive relations, only the relevant components need to be specified within a rule, which results in a high reuseability of the Deployment Enforcement Rules. Furthermore, we showed that the approach enables to define Whitelisting Rules, which specify what is allowed as well as Blacklisting Rules, which specify what is forbidden. Thus, depending on the requirements and circumstances the rules can be used very flexible. A validation of our approach is provided by a prototypical TOSCA-based implementation. In the future, we plan to extend our Deployment Enforcement Rules approach by also taking other TOSCA-elements into account, e.g., Deployment Artifacts attached to Node Templates.

Acknowledgment

This work was partially funded by the project SePiA.Pro (01MD16013F) of the BMWi program Smart Service World and the German Research Foundation (DFG) project ADDCompliance (314720630).

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," Computer Networks, vol. 54, no. 15, 2010, pp. 2787–2805.

[2] M. Hermann, T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios," in Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS). IEEE, 2016, pp. 3928–3937.

[3] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach," Transactions on Industrial Informatics, vol. 11, no. 3, 2015, pp. 812–820.

[4] M. Falkenthal et al., "Towards Function and Data Shipping in Manufacturing Environments: How Cloud Technologies leverage the 4th Industrial Revolution," in Proceedings of the 10th Advanced Summer School on Service Oriented Computing, ser. IBM Research Report. IBM Research Report, Sep. 2016, pp. 16–25.

[5] M. Zimmermann, U. Breitenbücher, M. Falkenthal, F. Leymann, and K. Saatkamp, "Standards-based function shipping how to use tosca for shipping and executing data analytics software in remote manufacturing environments," in Proceedings of the 2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC 2017), 2017, pp. 50–60.

[6] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and M. Wieland, "Policy-Aware Provisioning of Cloud Applications," in Proceedings of the Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2013). Xpert Publishing Services, Aug. 2013, pp. 86–95.

[7] U. Breitenbücher et al., "Policy-Aware Provisioning and Management of Cloud Applications," International Journal On Advances in Security, vol. 7, no. 1&2, 2014, pp. 15–36.

[8] OASIS, Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[9] T. Binz et al., "OpenTOSCA – A Runtime for TOSCA-based Cloud Applications," in Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). Springer, Dec. 2013, pp. 692–695.

[10] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann, "Winery – A Modeling Tool for TOSCA-based Cloud Applications," in Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). Springer, Dec. 2013, pp. 700–704.

[11] OASIS, TOSCA Simple Profile in YAML Version 1.0, Organization for the Advancement of Structured Information Standards (OASIS), 2015.

[12] OASIS, Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[13] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, TOSCA: Portable Automated Deployment and Management of Cloud Applications, ser. Advanced Web Services. Springer, Jan. 2014, pp. 527–549.

[14] Red Hat, Inc., "Ansible Official Site." [Online]. Available: https://www.ansible.com [retrieved: July, 2018]

[15] Opscode, Inc., "Chef Official Site." [Online]. Available: http://www.opscode.com/chef [retrieved: July, 2018]

[16] OASIS, Web Services Business Process Execution Language (WS-BPEL) Version 2.0, Organization for the Advancement of Structured Information Standards (OASIS), 2007.

[17] OMG, Business Process Model and Notation (BPMN) Version 2.0, Object Management Group (OMG), 2011.

[18] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann, "BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications," in Proceedings of the 4th International Workshop on the Business Process Model and Notation (BPMN 2012). Springer, Sep. 2012, pp. 38–52.

[19] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann, and T. Michelbach, "A Domain-Specific Modeling Tool to Model Management Plans for Composite Applications," in Proceedings of the 7th Central European Workshop on Services and their Composition, ZEUS 2015. CEUR Workshop Proceedings, May 2015, pp. 51–54.

[20] U. Breitenbücher et al., "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA," in International Conference on Cloud Engineering (IC2E 2014). IEEE, Mar. 2014, pp. 87–96.

[21] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Vinothek - A Self-Service Portal for TOSCA," in Proceedings of the 6th Central-European Workshop on Services and their Composition (ZEUS 2014). CEUR-WS.org, Feb. 2014, Demonstration, pp. 69–72.

[22] Eclipse Foundation, Inc., "Equinox — The Eclipse Foundation." [Online]. Available: http://www.eclipse.org/equinox/ [retrieved: July, 2018]

[23] S. Walraven, D. Van Landuyt, A. Rafique, B. Lagaisse, and W. Joosen, "Paashopper: Policy-driven middleware for multi-paas environments," Journal of Internet Services and Applications, vol. 6, no. 1, 2015, p. 2.

[24] K. Képes, U. Breitenbücher, M. P. Fischer, F. Leymann, and M. Zimmermann, "Policy-Aware Provisioning Plan Generation for TOSCA-based Applications," in Proceedings of The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE). XpertPublishing Services, September 2017, pp. 142–149.

[25] A. Blehm et al., "Policy-Framework-Eine Methode zur Umsetzung von Sicherheits-Policies im Cloud-Computing." in GI-Jahrestagung, 2014, pp. 277–288.

[26] T. Waizenegger et al., "Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing," in On the Move to Meaningful Internet Systems: OTM 2013 Conferences. Springer, Sep. 2013, pp. 360–376.

[27] M. P. Fischer, U. Breitenbücher, K. Képes, and F. Leymann, "Towards an Approach for Automatically Checking Compliance Rules in Deployment Models," in Proceedings of The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE). Xpert Publishing Services (XPS), 2017, pp. 150–153.

[28] M. Fellmann and A. Zasada, "State-of-the-art of business process compliance approaches," in 22nd European Conference on Information Systems, (ECIS), June 2014, pp. 1–17.

[29] F. Koetter, M. Kochanowski, A. Weisbecker, C. Fehling, and F. Leymann, "Integrating Compliance Requirements across Business and IT," in Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International. IEEE, 2014, pp. 218–225.

[30] D. Schleicher et al., "Compliance Domains: A Means to Model Data-Restrictions in Cloud Environments," in Enterprise Distributed Object Computing Conference (EDOC), 2011 15th European Conference on Information Systems IEEE International. IEEE, 2011, pp. 257–266.