TOSCA Light: Bridging the Gap Between the TOSCA Specification and Production-Ready Deployment Technologies

Michael Wurster[1], Uwe Breitenbücher[1], Lukas Harzenetter[1],
Frank Leymann[1], Jacopo Soldani[2] and Vladimir Yussupov[1]
[1] *Institute of Architecture of Application Systems, University of Stuttgart, Germany*
[2] *Department of Computer Science, University of Pisa, Pisa, Italy*
[lastname]@iaas.uni-stuttgart.de, [lastname]@di.unipi.it

Keywords: Deployment Automation, Cloud Computing, TOSCA.

Abstract: The automation of application deployment is critical because manually deploying applications is time-consuming, tedious, and error-prone. Several deployment automation technologies have been developed in recent years employing tool-specific deployment modeling languages. At the same time, the OASIS standard Topology Orchestration Specification for Cloud Applications (TOSCA) emerged as a means for describing cloud applications, i. e., their components and relationships, in a vendor-agnostic fashion. Despite TOSCA is widely used in research, it is not supported by the production-ready deployment automation technologies daily used by practitioners working with cloud-native applications, hence resulting in a gap between the state-of-the-art in research and state-of-practice in the industry. To help bridging this gap, we leverage the recently introduced Essential Deployment Metamodel (EDMM) and identify *TOSCA Light*, an EDMM-compliant subset of TOSCA, to enact the transformation from TOSCA to the vast majority of deployment automation technology-specific models used by today's software industry. Further, we present an end-to-end *TOSCA Light* modeling and transformation workflow and show a prototypical implementation to validate our approach.

1 INTRODUCTION

Modern software engineering industry focuses heavily on strengthening the synergy between development and operation processes, commonly referred to as *DevOps* (Humble and Molesky, 2011). One of the most important approaches is *Infrastructure as Code* (IaC), which allows application developers and operation engineers to specify the infrastructure of to-be-deployed applications "as code", which can then be processed by deployment automation technologies such as Chef, Puppet, or Terraform.

Deployment automation technologies typically rely on tool-specific declarative or imperative modeling languages. Declarative languages allow to describe the "what", i. e., the desired target state of the components to be deployed, while imperative languages allow indicating the "how", i. e., all technical tasks to be executed for deploying and configuring an application (Endres et al., 2017). Even if declarative languages are by far considered to be the most appropriate in practice (Wurster et al., 2019b), they are tightly coupled to the corresponding deployment automation technology, hence not favoring the portability of specified application deployments from one deployment automation technology to another.

In contrast, the *Topology and Orchestration Specification for Cloud Applications* (TOSCA) (OASIS, 2019) is a standardized cloud modeling language (CML) which allows to declaratively specify application deployments in a portable way. While TOSCA is heavily used in research (Bellendorf and Mann, 2019), it is currently not supported by the production-ready deployment automation technologies used by practitioners. As a result, a gap between the state-of-the-art in research and industry is arising.

One possible way to bridge this gap is to leverage the recently introduced Essential Deployment Metamodel (EDMM), which can be used to enable standards-based CMLs (like TOSCA) to get supported by existing deployment automation tooling. EDMM describes a set of core deployment modeling entities that the vast majority of deployment automation technologies understand (Wurster et al., 2019b). As a result, deployment models relying on the modeling constructs specified in the EDMM are easily convertible to multiple heterogeneous deployment automation formats, which is also shown by existing work (Wurster et al., 2019a). However, for mapping it to EDMM, and for seamlessly exploiting the ease-of-transformation of EDMM models, a CML has to comply with the set of requirements imposed by EDMM.

In this work, we present *TOSCA Light*, as an EDMM-compliant subset of TOSCA. TOSCA Light can be used for devising technology-agnostic application specifications that can be translated to technology-specific deployment artifacts, hence exploiting TOSCA to avoid coupling application deployments with particular technologies, while enabling their deployment on production-ready deployment technologies at the same time. Thus, TOSCA Light enhances the portability of models, requiring to model application deployments only once, and then allowing them to be reused for deploying applications with different, heterogeneous technologies. The main contributions of this paper are the following:

- We define the *TOSCA Light* subset of TOSCA, by identifying the set of EDMM compliance rules for TOSCA-based models.
- We describe an end-to-end *TOSCA Light* modeling and transformation workflow.
- We illustrate a proof-of-concept implementation of the *TOSCA Light* toolchain.

The paper is organized as follows. Section 2 provides some necessary background. Section 3 presents the TOSCA Light. Section 4 illustrates the end-to-end TOSCA Light modeling and transformation workflow, which prototypical implementation is shown in Sect. 5. Finally, Sect. 6 discusses related work and Sect. 7 draws concluding remarks.

2 BACKGROUND

This section presents the fundamentals and terminology needed in the rest of the paper.

2.1 TOSCA

TOSCA is an OASIS standard that enables modeling, provisioning, and management of cloud applications (OASIS, 2019; Binz et al., 2012). TOSCA allows to model so-called *Service Templates* that describe the topology, i. e., the components and their relationships, of an application to be deployed to a cloud infrastructure. The core modeling concepts in TOSCA are nodes and relationships. So-called *Node Templates* represent components of an application, such as virtual machines, web servers, or arbitrary software components, and so-called *Relationship Templates* represent the relations between those nodes, e. g., that a node is hosted on another node, or that a node connects to another node. Node and Relationship Templates are typed using *Node Types* and *Relationship Types* defining their semantics, and structuring them by listing their properties, attributes, requirements, capabilities, and interfaces.

Properties and attributes are used to configure the deployment, e. g., a "Tomcat" web server type may specify the property "port", which is filled with concrete values in the node template upon deployment. In the TOSCA metamodel, nodes get related to each other when one node has a requirement against some capability provided by another node. For example, a virtual machine node may offer the capability that a Tomcat web server node can be hosted on it. Further, TOSCA standardizes the so-called *Lifecycle Interface* specifying that node types may have "create", "configure", "start", "stop", and "delete" operations to be used for installing, configuring, starting, and stopping them. Such operations are implemented by so-called *Implementation Artifacts*, e. g., in the form of executable Shell scripts. In contrast, *Deployment Artifacts* implement a nodes' business logic. For example, the compressed binary files to run a Tomcat web server are meant to be attached as a deployment artifact to the respective type.

On top of that, TOSCA uses the notion of *Policies* to express non-functional requirements affecting an application topology at a certain stage of its lifecycle, and which are attached to single or groups of nodes in the application topology. For example, security aspects that need to be enforced at certain stage can be expressed via TOSCA policies (Yussupov et al., 2019). Beside *input* and *output parameters* to parameterize a service template, TOSCA defines the CSAR packaging format allowing to exchange applications.

2.2 Essential Deployment Metamodel

In recent years, many deployment technologies evolved following a declarative approach to automate the delivery of software components. Even if such technologies share the same purpose, they differ in features and supported mechanisms which is why comparing and selecting deployment automation technologies is difficult.

Recently, the Essential Deployment Metamodel (EDMM) was introduced as the result of a systematic review of deployment automation technologies (Wurster et al., 2019b). In this work, the authors extracted the essential modeling entities that are supported by the vast majority of declarative deployment automation technologies. The EDMM enables a common understanding of declarative deployment models and, thus, eases the comparison and selection of appropriate technologies. Figure 1 depicts the essential modeling entities of the normalized metamodel. EDMM defines *Components* as physical, functional,

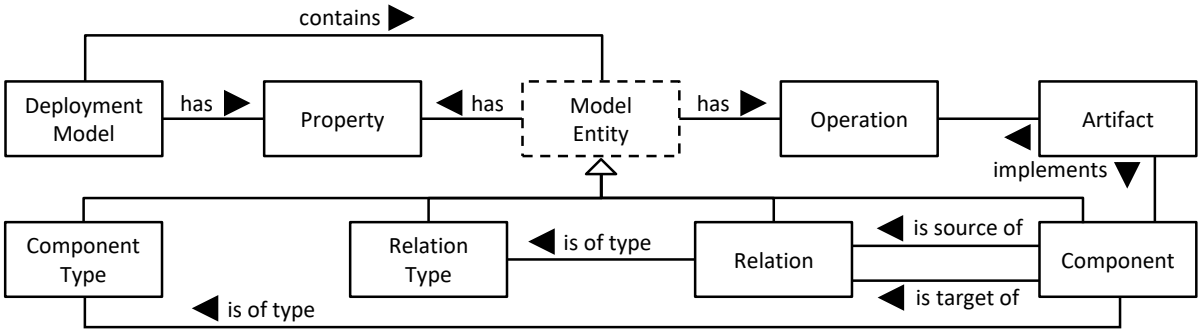

Figure 1: Essential Deployment Metamodel (EDMM) [adapted from Wurster et al. (2019b)].

or logical units of an application. Further, *Relations* are defined as directed physical, functional, or logical dependencies between exactly two components. Both can be typed using *Component Types* and *Relation Types* to express reusable entities that specify a certain semantic. Further, EDMM defines *Properties* as a way to describe the current state or prescribe the desired target state or configuration of a component or relation. Moreover, *Operations* are used in declarative deployment models to define executable procedures performed to manage a component or relation. Such operations provide hook points and are executed by deployment technologies to implement certain requirements during application deployment. the EDMM also defines *Artifacts* such that an artifact implements a component or operation and is therefore required for the execution of the application deployment as well as the final application system. Finally, according to EDMM, a *Deployment Model* describes the desired target state of an application including all necessary components, relations, properties, operations, and artifacts in a declarative manner.

3 TOSCA LIGHT

Recent work showed that an EDMM deployment model can be transformed into 13 concrete deployment automation technologies, such as Terraform and AWS CloudFormation (Wurster et al., 2019a). Thus, by reducing TOSCA to EDMM's essential modeling entities, which we call *TOSCA Light* hereafter, we enable the transformation of *TOSCA Light*-compliant models into the deployment technologies that are currently supported by the EDMM Transformation Framework. In the following, we present the TOSCA Light modeling profile which semantically maps to the EDMM modeling constructs introduced in the previous section. We show the supported TOSCA Light modeling profile based on TOSCA's YAML grammar for describing service templates, type definitions, topology templates, their nodes, relationships, and properties (OASIS, 2019). Further, for the sake of clarity, we show several TOSCA Light examples and code snippets.

3.1 Service Template

A *Service Template* in TOSCA is used to define the topology of application components and cloud services so that they can be deployed in accordance with constraints and policies. In particular, service templates allow the definitions of a topology template, types (e. g., node, relationship, capability, artifact types), groupings, policies, and constraints along with any input or output parameters (cf. Sect. 2). However, not all of these entities are compliant to EDMM and, therefore, not part of TOSCA Light. In general, the *Service Template* entity of TOSCA's metamodel corresponds to EDMM's *Deployment Model* as it describes declaratively the desired target state of an application including all necessary model entities. Listing 1 shows the service template definition according to TOSCA Light. It can include arbitrary node types, relationship types and interface types, provided

```
tosca_definitions_version: tosca_light_yaml_1_0
metadata:
  # map of key/value pairs
description: <value>
node_types:
  # map of TOSCA Light node type definitions
relationship_types:
  # map of TOSCA Light relationship types
interface_types:
  # map of TOSCA Light interface types
topology_template:
  # TOSCA Light topology template definition
```

Listing 1: TOSCA Light service template.

that such definition are *TOSCA Light*-compliant (cf. Sect. 3.2), as well as *TOSCA Light*-compliant topology template definition (cf. Sect. 3.3). Moreover, *description* and *metadata* information can be defined to convey additional information not required for the actual deployment automation.

Condition 1: A *TOSCA Light*-compliant service template can *only* contain what follows:

- ➔ Description and additional metadata information, with the latter given as a map of key/value pairs.
- ➔ Definitions of reusable *TOSCA Light*-compliant node types.
- ➔ Definitions of reusable *TOSCA Light*-compliant relationship types.
- ➔ Definitions of reusable *TOSCA Light*-compliant Light interface types.

3.2 Type Definitions

TOSCA defines type entities, such as group types, data types, capability types, and policy types. EDMM generally does not define metamodel entities that map to such TOSCA entities (cf. Fig. 1). Therefore, these types cannot be semantically mapped to EDMM and are not supported by TOSCA Light.

Further, TOSCA Light limits the use of interface types. Wurster et al. (2019b) defined that EDMM allows to hook into or influence the deployment lifecycle. This means that application developers must be able to specify operations along the application's deployment lifecycle, e. g., to install, start, stop, or terminate components. As TOSCA standardizes the *Lifecycle Interface* and defines the essential operations that must be supported, such as starting, configuring, and stopping a component, the map of possible TOSCA Light interface types is restricted to this single type. Hence, models defining custom interface types are not supported by TOSCA Light.

Weerasiri et al. (2017) defined in their taxonomy of cloud resource orchestration techniques that it must be possible to express such that (i) a component generally depends on another component, (ii) a component is "hosted on" or "contained in" another component, and (iii) a component connects to another component. Therefore, TOSCA Light limits the support of relationship types to "DependsOn", "HostedOn", and "ConnectsTo" as defined accordingly in the normative type reference of the TOSCA Simple Profile specification (OASIS, 2019). Hence, custom relationship types must inherit from these types, otherwise, the model is not compliant with TOSCA Light.

Semantically, TOSCA node and relationship types refer to EDMM's *Component Type* and *Relationship Type* metamodel entities. Such types specify the semantics by means of properties and operations, and are often meant to be reused in multiple deployment models. The TOSCA Light modeling profile allows the definition of `attributes` and `properties` on such types, as shown in Listing 2 on the example of node types. These are defined by a type (in accordance to the YAML 1.2 spec), a description, an optional default value, and a required flag, and correspond to the *Property* entity in EDMM.

```
tosca_definitions_version: tosca_light_yaml_1_0
node_types:
  <node_type_name>:
    derived_from: <parent_node_type_name>
    metadata:
      # map of key/value pairs
    description: <value>
    attributes:
      # map of TOSCA Light attribute definitions
    properties:
      # map of TOSCA Light property definitions
    interfaces:
      # map of TOSCA Light interface definition
    artifacts:
      # map of TOSCA Light artifact definitions
```

Listing 2: TOSCA Light node type.

Apart from a "description" and additional "metadata" information, the map of artifacts corresponds to the *Artifact* entity in EDMM. The map defines a named list of files that are associated with this type and used by deployment automation technologies to facilitate the deployment and implementation of interface operations. To hook into or influence the deployment lifecycle, a TOSCA Light node type may define several operations based on TOSCA's standardized lifecycle interface. Such operations correspond to EDMM's *Operation* entity and define possible executable procedures performed to deploy this type.

```
tosca_definitions_version: tosca_light_yaml_1_0
topology_template:
  inputs:
    # map of TOSCA Light property definitions
  outputs:
    # map of TOSCA Light property definitions
  node_templates:
    # map of TOSCA Light node templates
  relationship_templates:
    # map of TOSCA Light relationship templates
```

Listing 3: TOSCA Light topology template.

Condition 2: A specification is *TOSCA Light*-compliant if the following *constraints on types* hold:

- ➔ The only employed interface type is the TOSCA's standard lifecycle interface.
- ➔ Employed relationship types are restricted to TOSCA's normative types "DependsOn", "HostedOn", and "ConnectsTo", and to custom types defined by extending such types.
- ➔ Node types and relationship types can define a description and additional metadata information, with the latter given as a map of key/value pairs.
- ➔ If node types and relationship types define "attributes" and "properties", these specify a type, description, default value, and required flag.
- ➔ If node types and relationship types define operations, these are based on TOSCA's standardized lifecycle interface.
- ➔ If node types and relationship types list "artifacts", these are used to implement nodes, relations, or interface operations.

3.3 Topology Template

A *Topology Template* defines the overall structure of an application deployment and is part of a service template. It contains the set of *Node Template* and *Relationship Template* definitions that together define the topology of the application as a directed graph, which in turn correspond to the *Components* and *Relations* entities in EDMM. Further, TOSCA Light allows to define input and output parameters to parameterize a topology model instead of using fixed values only. Such parameters correspond to the *Property* entity in EDMM and can be defined by a type (in accordance to the YAML 1.2 spec), a description, an optional default value or a computed value by a composition of node properties. The "policy" and "workflow" construct defined by TOSCA (OASIS, 2019) is not supported by TOSCA Light as there is no corresponding entity in EDMM.

Listing 4 shows a simple TOSCA Light example to install a MySQL database system on top of an Ubuntu virtual machine. The Ubuntu virtual machine and the MySQL software component are expressed as separate node templates and specify the occurrences of components as part of the deployment model. The example shows how input parameters are used to parameterize the definition of the used root password. Further, a "configure" script is assigned as an operation to execute custom logic during deployment.

Even more, the example highlights that TOSCA in general relies on capabilities and requirements. For example, capabilities are used to define additional properties for a node, e. g., the type of operating system to be used. Further, TOSCA relies on matching capabilities and requirements to express certain relationships between nodes, e. g., the "hosted on" relationship between the nodes in Listing 4.

```
tosca_definitions_version: tosca_light_yaml_1_0
topology_template:
  inputs:
    password:
      type: string
    port:
      type: string
      default: 3306
  node_templates:
    mysql:
      type: tosca.nodes.DBMS.MySQL
      properties:
        root_password: { get_input: password }
        port: { get_input: port }
      interfaces:
        Standard:
          operations:
            configure: configure.sh
      requirements:
        - host:
            node: db_server
            relationship: hosted_on
    db_server:
      type: tosca.nodes.Compute
      capabilities:
        os:
          properties:
            architecture: x86_64
            type: linux
            distribution: ubuntu
            version: 18.04 LTS
  relationship_templates:
    hosted_on:
      type: tosca.relationships.HostedOn
```

Listing 4: TOSCA Light template to install MySQL.

However, TOSCA Light is not able to use requirements and capabilities as a first-class modeling construct as there is no matching entity available in EDMM. Instead, TOSCA Light relies on the defined normative requirements and capabilities of the TOSCA Simple Profile (OASIS, 2019) and maps them semantically to properties (cf. operating system properties) or considers them as a relationship definition (cf. hosted on relationship). Moreover, advanced TOSCA features such as "substitution mappings" or the `node_filter` setting are in general not supported by TOSCA Light. Anyhow, a TOSCA modeling tool that supports both, TOSCA and TOSCA Light, could try to resolve open requirements, resolve substitutable types, and resolve the `node_filter` setting based on a given TOSCA type repository before further processing it as a TOSCA Light model.

Condition 3: A *TOSCA Light*-compliant topology template can *only* contain what follows:

- ➔ Definition of "input" and "output" parameters, each specifying a type, description, value, or computed value by a composition of properties.
- ➔ Node templates declaratively specifying the components to be deployed.
- ➔ Relationship templates indicating the functional or logical dependency between two nodes.
- ➔ Concrete values assigned to defined "properties".
- ➔ Additional operation definitions, still based on TOSCA's normative lifecycle interface.
- ➔ Usage of TOSCA's normative "capabilities".

3.4 Defining TOSCA Light

In summary, a TOSCA application has to satisfy conditions 1, 2, and 3 respectively discussed in Sect. 3.1 to 3.3 to be compliant with TOSCA Light.

Condition 1 defines that a TOSCA Light service template, among the description and additional descriptive metadata information, defines reusable sets of TOSCA Light compliant node types, relationship types, and interface types.

Condition 2 restricts type definitions. TOSCA Light only allows the usage of the standard lifecycle interface and the normative relationship types defined by the TOSCA standard. Hence, custom relationship types must inherit from the normative types. Further, the definition of "operations" on node types and relationship types must be based on the standard lifecycle interface. Along with description and metadata, node types and relationship types may define "attributes" and "properties" to describe the current state or desired target state of configuration. Finally, node types and relationship types may define "artifacts" to implement nodes, relations, or interface operations.

Condition 3 defines additional restrictions on topology templates. TOSCA Light compliant topology templates may define "input" and "output" parameters. Further, a set of node templates and relationship templates define the declarative specification of components to be deployed and the functional or logical dependency between them. Moreover, node templates and relationship templates may assign concrete values to defined "properties" as well as define additional "artifacts" and "operations" as long as they are comply with TOSCA's normative lifecycle interface. TOSCA Light supports the use of TOSCA's normative "requirements" and "capabilities" to establish relationships or define additional properties. However, the definition of *open* requirements (e.g., `node_filter`) is not supported.

4 END-TO-END TOOLCHAIN

In this section, we elaborate on the end-to-end modeling and transformation workflow using TOSCA Light. The overall process comprises four steps as shown in Fig. 2 and starts with the creation of an EDMM-compliant TOSCA Light deployment model.

4.1 Create Deployment Model

An application component architecture is specified using the TOSCA CML to produce deployment model.

Workflow. The overall modeling workflow, essentially, does not differ from the common process. A given application's component architecture is described declaratively using TOSCA constructs either using a graphical modeling tool or a text editor. The resulting set of modeling artifacts comprises the TOSCA-based definitions describing types and component instances as well as the file artifacts required for deploying the application, e. g., container images.
Tooling. The deployment model can be specified using multiple possible tools, including graphical modeling tools such as Eclipse Winery (Kopp et al., 2013) or Alien4Cloud (ALIEN 4 Cloud, 2020). Another possible option is to use text-based editors such as Visual Studio Code offered by Microsoft.

4.2 Validate TOSCA Light Compliance

The produced TOSCA-based deployment model is analyzed to ensure its compliance to the TOSCA Light modeling profile and, hence, to EDMM.

Workflow. As a next step, the resulting TOSCA-based deployment model is validated for EDMM compliance. To achieve this, the given model is checked against the set of modeling requirements discussed in-detail in Sect. 3. In case the model is EDMM-compliant it is labeled as TOSCA Light and can be used as an input for transforming the model into the desired target deployment format such as Terraform or Kubernetes. In situations when the model is not compatible with the TOSCA Light modeling profile, the model can be refined according to the provided modification recommendations.
Tooling. Essentially, the validation of the model can be performed either at design time using graphical modeling tools or text editors, or at transformation time using transformation tools. In case of the former, corresponding TOSCA Light compliance plugins have to be introduced. Further, tools like Eclipse

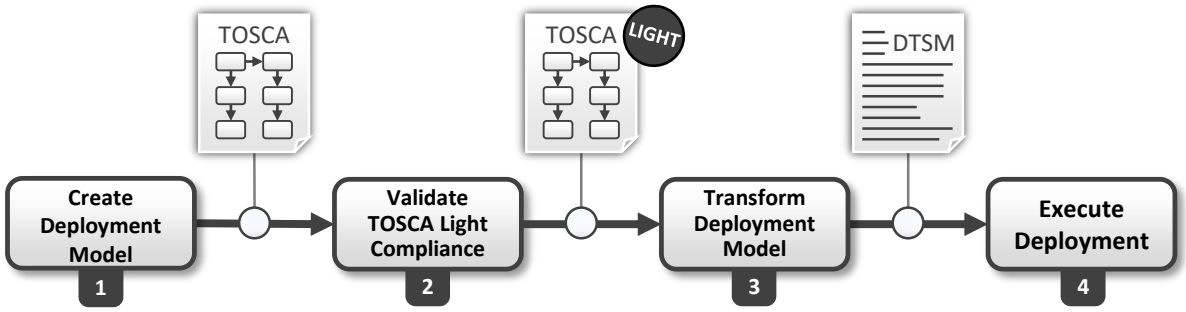

Figure 2: TOSCA Light end-to-end toolchain.

Winery could also provide a TOSCA Light compliance check while importing existing TOSCA applications or by providing mechanisms to verify a given TOSCA repository regarding compliance.

4.3 Transform Deployment Model

The successfully validated TOSCA Light model is transformed into the target deployment technology.

Workflow. Afterwards, the successfully validated TOSCA Light model is used as an input for the transformation engine to produce a deployment technology-specific model (DTSM). Consequently, another input required for the transformation engine is the name of the target deployment automation tool, e. g., Ansible or Terraform. As discussed previously, the possibility to transform the model is guaranteed by design since the used modeling constructs conform to the essential entities defined by EDMM.
Tooling. The EDMM Transformation Framework is used to generate the DTSM (Wurster et al., 2019a). To achieve this, a given TOSCA Light model's constructs are mapped to the corresponding entities in the target tool's constructs space. It is worth mentioning, that the generated target model might be refined with, e. g., security credentials or other configuration information in the form variables that are relevant only for the target deployment automation technology.

4.4 Execute Deployment

The transformed technology-specific deployment model is executed using the standard mechanisms provided by the target deployment automation tool.

Workflow. Finally, after the generated target deployment model is ready to be deployed, it can be used as an input for the target deployment automation technology. Hence, the actual deployment happens using the standard tool's mechanisms, which also simplifies further management of the application.
Tooling. Since the deployment happens using the standard tool-specific deployment mechanisms, the actual tooling that needs to be used depends on the concrete use case, i. e., which target deployment technology was used. This also influences the way the tool has to be used, i. e., where it can be hosted or who operates it. For example, the interaction will be different for *as-a-service* solutions such as AWS CloudFormation and self-hosted tools such as Ansible.

5 PROOF-OF-CONCEPT IMPLEMENTATION

In this section, we illustrate a proof-of-concept (PoC) implementation based on (i) Eclipse Winery (Kopp et al., 2013) and (ii) the EDMM Transformation Framework[1] (Wurster et al., 2019a) to validate the feasibility of the proposed TOSCA Light end-to-end toolchain. Next, we briefly explain the extensions, the usage of the tools, and how they map to the proposed end-to-end toolchain in Sect. 4.

Eclipse Winery is a web-based environment to graphically model TOSCA-based application topologies. It provides a *Management Interface* to manage all TOSCA related entities, such as node types, their property definitions, operations, and artifacts. Further, Winery provides a *Topology Modeler* component which enables the graphical composition of the desired target state of application to be deployed. For this PoC, Winery has been extended by the *TOSCA Light Mode* and is used to define and develop the deployment model graphically in accordance to step 1 in Fig. 2. We created a fork of Eclipse Winery and published our extension to GitHub[2] respectively. The current implementation checks the TOSCA Light com-

[1] https://bit.ly/2WzlicC
[2] https://bit.ly/2xg4hK0

pliance when a user opens a TOSCA service template, which implements step 2 of the proposed TOSCA Light toolchain (cf. Fig. 2). Each created or imported TOSCA model may be flagged as TOSCA Light compliant by showing a TOSCA Light logo in Winery's header component. Further, Winery is able to provide a list of violated conditions when a TOSCA service template is not compliant with TOSCA Light.

In terms of transformation (cf. step 3 in Fig. 2), we exploit the existing EDMM export functionality by Eclipse Winery, as TOSCA Light is fully compliant to EDMM. The exported EDMM model can then directly be processed by the the EDMM Transformation Framework (Wurster et al., 2019a), either by using the provided CLI or the REST interface over HTTP. Together with the EDMM model, a user can select a certain target deployment technology and the framework generates the respective files and templates. Finally, and according to step 4 (cf. Fig. 2), the output can be executed using the target technology's tooling.

6 RELATED WORK

The OASIS standard TOSCA constitutes a reference metamodel for specifying the topology and orchestration of multi-component cloud applications (Bergmayr et al., 2018). At the same time, most existing cloud deployment automation technologies are not offering a native support to TOSCA, which makes it complex to deploy and manage TOSCA-based application in production-ready environments.

Currently existing efforts for enacting the deployment of TOSCA-based applications can be clustered in three main categories (Bellendorf and Mann, 2019). We can indeed distinguish (i) solutions for *directly deploying* TOSCA application specifications in cloud infrastructure, (ii) approaches *integrating TOSCA with other standards* for enhancing deployment automation, and (iii) solutions for enabling the deployment of TOSCA-based applications on *existing deployment technologies*.

The reference approach for directly deploying TOSCA-based cloud application is the OpenTOSCA engine proposed by Binz et al. (2013), recently extended to also support the deployment of applications also in IoT environments (da Silva et al., 2016). The OpenTOSCA engine enables the orchestration of the deployment and management of TOSCA-based applications on a target infrastructure, by obviously requiring to get installed in the manager of such infrastructure and of being provided with the TOSCA specifications of to-be-orchestrated applications. It is however intended to be itself the orchestrator of the application, and it currently does not support streamlining the deployment and management of an application to other existing deployment technologies (e. g., Kubernetes or Terraform), which are currently more used in production-ready environments (Pahl et al., 2019).

Similar considerations apply to other existing approaches enabling the deployment and management of TOSCA-based application on target infrastructures, e. g., the DevOps-based streamlining proposed by Wettinger et al. (2014, 2015), the TOSCA support provided by Cloudify (Cloudify Platform Ltd., 2020), the open-source initiatives AriaTOSCA (Apache Software Foundation, 2018) and Alien4Cloud (ALIEN 4 Cloud, 2020), and the multi-cloud orchestration enacted by SeaClouds (Brogi et al., 2016). All such approaches rely on the availability of full-fledged TOSCA-compliant orchestrators, while our objective is to identify the subset of TOSCA enabling the specification multi-component applications that can then be processed by most used production-ready cloud deployment automation technologies. We enable this by identifying the subset of TOSCA that is compliant with the EDMM proposed by Wurster et al. (2019b), and by providing the tooling needed to enact the translation of TOSCA Light application specifications on all cloud deployment automation technologies supported by the EDMM Transformation Framework (Wurster et al., 2019a).

Other approaches worth relating to ours are those integrating TOSCA with other standards for enhancing deployment automation. Noteworthy efforts in this direction are those by Calcaterra et al. (2017, 2018a) and by Kopp et al. (2012), both working on the integration of BPMN with TOSCA to imperatively program and automate the deployment and management of multi-cloud application. Calcaterra et al. (2017, 2018a) propose a solution to automatically generate BPMN workflows for deploying and managing multi-component applications, starting from their TOSCA specification. They also provide an extension of their approach, which allows to deal with potential failures while actually enacting the generated BPMN workflows (Calcaterra et al., 2018b). Kopp et al. (2012) instead proposes a BPMN profile for enabling the imperative programming of the deployment and management of multi-component applications specified in TOSCA. Hence, even if different in the spirit, both the approaches by Calcaterra et al. (2017, 2018a) and by Kopp et al. (2012) enable automating the deployment and management of TOSCA applications by relying on the availability of a BPMN engine.

Cardoso et al. (2013) and Glaser. et al. (2017) are worth mentioning efforts integrating TOSCA with other standards for enhancing deployment automa-

tion. Cardoso et al. (2013) propose to integrate TOSCA with the service description language USDL, by exploiting the latter to define the functionalities of cloud services, and by providing a prototypical platform integrating service selection with deployment. Glaser. et al. (2017) instead provide analyze the analogies and possible integration between TOSCA and OCCI (Open Grid Forum, 2016). They then propose a fully-automated model-driven cloud application orchestrator based on the two standards.

However, the approaches by Calcaterra et al. (2017, 2018a), Kopp et al. (2012), Cardoso et al. (2013), and Bergmayr et al. (2016) all aim at tackling objectives quite different from ours. They succeed in enabling the deployment and management of TOSCA-based applications based on the integration of TOSCA with existing standards, but they are not supporting the possibility of enacting the deployment of TOSCA-based applications using production-ready deployment automation technologies.

Last, but probably the most related to our approach, are the solutions enabling the deployment of TOSCA-based applications on existing cloud deployment automation technologies. Breiter et al. (2014) illustrate how to deploy multi-component applications specified in TOSCA on the IBM cloud computing infrastructure. Brogi et al. (2018b,a) propose a TOSCA profile for specifying container-based multi-component application, by also providing the TosKer engine for actually enacting their deployment and management of such application on Docker-enabled infrastructures. Carrasco et al. (2016); Carrasco et al. (2018) enable trans-cloud application deployment, by allowing to run TOSCA application specifications on top of Apache Brooklyn (Apache Software Foundation, 2020). However, all such efforts target a precise infrastructure or a precise cloud deployment automation technology. We instead aim at enabling the deployment of TOSCA applications with most used production-ready deployment technologies.

Similar considerations apply to the noteworthy efforts by Katsaros et al. (2014) and Tricomi et al. (2017). Both proposing different approaches for enabling the deployment of multi-component applications specified in TOSCA on OpenStack cloud infrastructures. Gusev et al. (2014) and Ivanovska et al. (2015) goes even a step further by proposing the P-TOSCA environment, which enables multi-cloud deployment and management of TOSCA-based applications on both Eucalyptus and OpenStack.

In summary, to the best of our knowledge, all existing approaches for enabling the deployment of TOSCA application focus on supporting the full expressiveness of TOSCA, either by requiring to process application specifications with TOSCA-compliant engines or by requiring to run them with some specific third-party orchestrator. Our aim is instead to identify the subset of TOSCA (i. e., the TOSCA Light profile) that can be processed by any production-ready cloud deployment automation technology, to give application administrators the freedom of choosing the technology most suited to their needs. We indeed identify the subset of TOSCA that is compliant with the EDMM metamodel, which is known to distill the essentials of cloud deployment automation technologies (Wurster et al., 2019b). Further, we provide all tooling needed to actually translate TOSCA Light application specifications into deployment technology-specific models (DTSM) allowing to enact their deployment and management on production-ready deployment automation technologies, i. e., those currently supported by the EDMM Modeling and Transformation Framework (Wurster et al., 2019a).

7 CONCLUSIONS

TOSCA as a standardized cloud modeling language is indeed widely used in research. However, in industry, concrete deployment automation technologies are much more common as they often provide comprehensive tooling and support for integration in modern software development and operations processes. To close this gap, we identified and introduced the TOSCA Light modeling profile, a set of EDMM compatibility rules resulting in a reduced set of TOSCA modeling constructs, and described the TOSCA Light end-to-end toolchain. Further, we presented a PoC implementation of the end-to-end toolchain by extending Eclipse Winery to interactively check TOSCA applications for TOSCA Light compliance. By leveraging EDMM and its transformation capabilities, we enact to translate TOSCA-based application deployment models into concrete deployment technology-specific models used by popular deployment automation technologies.

For immediate future work, we plan to fully implement and evaluate the proposed TOSCA Light toolchain. The modeling complexity as well as the engineered prototype will be evaluated using a real-world industrial scenario. We plan to show a detailed study on how TOSCA Light can simplify the management of applications running on Kubernetes.

Beside of that, we plan to investigate the relation of EDMM to other cloud modeling languages (Bergmayr et al., 2018), such as CAMEL or CloudML, to enact the translation based on the EDMM Transformation Framework.

ACKNOWLEDGMENTS

This work is partially funded by the EU project *RADON* (825040), the German Research Foundation (DFG) project SustainLife (379522012), and the projects *AMaCA* (POR-FSE) and *DECLware* (University of Pisa, PRA_2018_66).

REFERENCES

ALIEN 4 Cloud (2020). ALIEN 4 Cloud. Version 3, https://alien4cloud.github.io.

Apache Software Foundation (2018). AriaTOSCA. Incubator, https://ariatosca.incubator.apache.org.

Apache Software Foundation (2020). Apache Brooklyn. Version 1, https://brooklyn.apache.org.

Bellendorf, J. and Mann, Z. A. (2019). Specification of cloud topologies and orchestration using TOSCA: A survey. *Computing*.

Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., and Leymann, F. (2018). A Systematic Review of Cloud Modeling Languages. *ACM Comput. Surv.*, 51(1).

Bergmayr, A., Breitenbücher, U., Kopp, O., Wimmer, M., Kappel, G., and Leymann, F. (2016). From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2*, CLOSER 2016, page 97–108. SciTePress.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). OpenTOSCA — A Runtime for TOSCA-Based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing - Volume 8274*, ICSOC 2013, page 692–695, Berlin, Heidelberg. Springer-Verlag.

Binz, T., Breiter, G., Leymann, F., and Spatzier, T. (2012). Portable Cloud Services Using TOSCA. *IEEE Internet Computing*, 16(03):80–85.

Breiter, G., Behrendt, M., Gupta, M., Moser, S. D., Schulze, R., Sippli, I., and Spatzier, T. (2014). Software defined environments based on TOSCA in IBM cloud implementations. *IBM Journal of Research and Development*, 58(2/3):9:1–9:10.

Brogi, A., Carrasco, J., Cubo, J., D'Andria, F., Di Nitto, E., Guerriero, M., Pérez, D., Pimentel, E., and Soldani, J. (2016). SeaClouds: An Open Reference Architecture for Multi-cloud Governance. In Tekinerdogan, B., Zdun, U., and Babar, A., editors, *Software Architecture*, volume 9839 of *LNCS*, pages 334–338. Springer International Publishing.

Brogi, A., Neri, D., Rinaldi, L., and Soldani, J. (2018a). Orchestrating incomplete TOSCA applications with Docker. *Science of Computer Programming*, 166:194–213.

Brogi, A., Rinaldi, L., and Soldani, J. (2018b). TosKer: A Synergy Between TOSCA and Docker for Orchestrating Multicomponent Applications. *Software: Practice and Experience*, 48(11):2061–2079.

Calcaterra, D., Cartelli, V., Di Modica, G., and Tomarchio, O. (2017). Combining TOSCA and BPMN to Enable Automated Cloud Service Provisioning. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science*, CLOSER 2017, page 187–196. SciTePress.

Calcaterra, D., Cartelli, V., Di Modica, G., and Tomarchio, O. (2018a). A Framework for the Orchestration and Provision of Cloud Services Based on TOSCA and BPMN. In Ferguson, D., Muñoz, V. M., Cardoso, J., Helfert, M., and Pahl, C., editors, *Cloud Computing and Service Science*, pages 262–285, Cham. Springer International Publishing.

Calcaterra, D., Cartelli, V., Modica, G. D., and Tomarchio, O. (2018b). Exploiting BPMN Features to Design a Fault-aware TOSCA Orchestrator. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, pages 533–540. SciTePress.

Cardoso, J., Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2013). Cloud Computing Automation: Integrating USDL and TOSCA. In Salinesi, C., Norrie, M. C., and Pastor, Ó., editors, *Advanced Information Systems Engineering*, pages 1–16. Springer Berlin Heidelberg.

Carrasco, J., Cubo, J., Durán, F., and Pimentel, E. (2016). Bidimensional Cross-Cloud Management with TOSCA and Brooklyn. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 951–955.

Carrasco, J., Durán, F., and Pimentel, E. (2018). Transcloud: CAMP/TOSCA-based bidimensional cross-cloud. *Computer Standards & Interfaces*, 58:167 – 179.

Cloudify Platform Ltd. (2020). TOSCA Orchestration & Training. https://cloudify.co/tosca.

da Silva, A. C. F., Breitenbücher, U., Képes, K., Kopp, O., and Leymann, F. (2016). OpenTOSCA for IoT: Automating the Deployment of IoT Applications Based on the Mosquitto Message Broker. In *Proceedings of the 6th International Conference on the Internet of Things*, page 181–182. ACM.

Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., and Wettinger, J. (2017). Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In *Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS)*, pages 22–27. Xpert Publishing Services.

Glaser., F., Erbel., J., and Grabowski., J. (2017). Model Driven Cloud Orchestration by Combining TOSCA and OCCI. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, pages 672–678. SciTePress.

Gusev, M., Kostoska, M., and Ristov, S. (2014). Cloud P-TOSCA porting of N-tier applications. In *2014 22nd*

Telecommunications Forum Telfor (TELFOR), pages 935–938.

Humble, J. and Molesky, J. (2011). Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, 24(8):6.

Ivanovska, B., Ristov, S., Kostoska, M., and Gusev, M. (2015). Using the P-TOSCA model for energy efficient cloud. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 245–249.

Katsaros, G., Menzel, M., Lenk, A., Revelant, J. R., Skipp, R., and Eberhardt, J. (2014). Cloud Application Portability with TOSCA, Chef and Openstack. In *2014 IEEE International Conference on Cloud Engineering*, pages 295–302.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2012). BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In Mendling, J. and Weidlich, M., editors, *Business Process Model and Notation*, volume 125 of *Lecture Notes in Business Information Processing*, pages 38–52. Springer Berlin Heidelberg.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2013). Winery – a modeling tool for tosca-based cloud applications. In *International Conference on Service-Oriented Computing*, pages 700–704. Springer.

OASIS (2019). TOSCA Simple Profile in YAML Version 1.3.

Open Grid Forum (2016). Open Cloud Computing Interface (OCCI). `https://occi-wg.org`.

Pahl, C., Brogi, A., Soldani, J., and Jamshidi, P. (2019). Cloud Container Technologies: A State-of-the-Art Review. *IEEE Transactions on Cloud Computing*, 7(3):677–692.

Tricomi, G., Panarello, A., Merlino, G., Longo, F., Bruneo, D., and Puliafito, A. (2017). Orchestrated Multi-Cloud Application Deployment in OpenStack with TOSCA. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6.

Weerasiri, D., Barukh, M. C., Benatallah, B., Sheng, Q. Z., and Ranjan, R. (2017). A Taxonomy and Survey of Cloud Resource Orchestration Techniques. *ACM Computer Surveys*, 50(2).

Wettinger, J., Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2015). *Streamlining Cloud Management Automation by Unifying the Invocation of Scripts and Services Based on TOSCA*, pages 2240–2261. IGI Global.

Wettinger, J., Binz, T., Breitenbücher, U., Kopp, O., Leymann, F., and Zimmermann, M. (2014). Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA. In *Proceedings of the 4th International Conference on Cloud Computing and Service Science, CLOSER 2014, 3-5 April 2014, Barcelona, Spain*, pages 559–568. SciTePress.

Wurster, M., Breitenbücher, U., Brogi, A., Falazi, G., Harzenetter, L., Leymann, F., Soldani, J., and Yussupov, V. (2019a). The EDMM Modeling and Transformation System. In *Service-Oriented Computing – ICSOC 2019 Workshops*. Springer.

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., and Soldani, J. (2019b). The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies. *SICS Software-Intensive Cyber-Physical Systems*.

Yussupov, V., Falazi, G., Falkenthal, M., and Leymann, F. (2019). Protecting Deployment Models in Collaborative Cloud Application Development. *International Journal On Advances in Security*, pages 79–94.