Technology-Agnostic Declarative Deployment Automation of Cloud Applications

Michael Wurster[1(✉)], Uwe Breitenbücher[1], Antonio Brogi[2], Lukas Harzenetter[1], Frank Leymann[1], and Jacopo Soldani[2]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{wurster,breitenbuecher,harzenetter,leymann}@iaas.uni-stuttgart.de

[2] Department of Computer Science, University of Pisa, Pisa, Italy
{brogi,soldani}@di.unipi.it

Abstract. Declarative approaches for automating the deployment and configuration management of multi-component applications are on the rise. Many deployment technologies exist, sharing the same baselines for enacting declarative deployments, even if based on different languages for specifying multi-component applications. The Essential Deployment Metamodel (EDMM) Modeling and Transformation Framework allows to specify multi-component applications in a technology-agnostic manner, and to automatically generate the technology-specific deployment artifacts allowing to deploy an IaaS-based application. In this paper, we propose an extension of the EDMM Modeling and Transformation Framework to PaaS and SaaS by allowing to deploy application components on PaaS platforms or to implement them by instrumenting SaaS services. Given that not all existing deployment technologies support PaaS and SaaS deployments, we also propose the new EDMM Decision Support Framework allowing us to determine which deployment technologies can be used to deploy an application specified with EDMM.

Keywords: Deployment modeling · Deployment automation · Cloud application

1 Introduction

The widespread of cloud computing and DevOps resulted in a plethora of different deployment technologies being proposed. These aim at establishing highly automated deployment processes, as manual deployments of complex multi-component applications is cumbersome and error-prone [18,25]. By describing the components and infrastructure of an application in reusable deployment models, a repeatable end-to-end deployment automation can be established.

This is typically done by following a declarative approach, i. e., by specifying the structure of an application and the desired state into which an application or parts thereof have to be transferred [15]. The declarative approach is indeed considered the most appropriate for application deployment and configuration management [6,17,31], as also witnessed by the multitude of existing deployment technologies following such an approach, e. g., AWS CloudFormation, Chef, Juju, Kubernetes, Puppet, and Terraform, just to mention some.

At the same time, existing declarative deployment technologies differ in supported features and mechanisms, as well as in the modeling language for describing the application and its desired state. Open standards (e. g., TOSCA [23,24]) have been proposed to ensure the portability of cloud application deployments from a provider/technology to another. However, major providers and deployment technologies are currently not supporting such standards. This makes it difficult to compare technologies based on their capabilities, select a deployment technology that is suited to accomplish given requirements, and to migrate a deployment model from one technology to another.

In our previous work, we tackled the aforementioned issue by starting from most used declarative deployment technologies and by distilling their essential parts into what we called the *Essential Deployment Metamodel* (EDMM) [31]. We also implemented a concrete YAML-based language for modeling applications with EDMM. Further, we proposed the *EDMM Modeling and Transformation Framework* [30] allowing to exploit EDMM as a "normalized metamodel" to deploy the same application with different technologies: After specifying the application with EDMM, the transformation framework can automatically generate the deployment artifacts needed to deploy the application with the selected target deployment technology. Notably, by simply re-running the transformation framework with a different target deployment technology, the same application specification can be used to migrate the deployment of an application from one technology to another [30].

The EDMM Modeling and Transformation Framework, however, currently supports the deployment of multi-component applications only on virtual compute resources such as virtual machines or containers (i. e., IaaS). In this paper, we overcome this limitation by providing the following two main contributions:

1. We extend the EDMM Modeling and Transformation Framework to deploy application components also on *PaaS platforms*, as well as to exploit existing *SaaS services* to implement components.
2. We present the *EDMM Decision Support Framework* allowing us to determine which declarative deployment technologies can be used to deploy a given EDMM model.

The latter is intended to help application developers to avoid trying to deploy an application with a deployment technology not offering the needed features, e. g., Juju is intended to automate the deployment of multi-component applications over IaaS-based virtual machines, but it cannot be used to deploy application components on PaaS platforms.

The rest of the paper is organized as follows. Section 2 presents the fundamentals and motivations for our work. Section 3 introduces our approach and Sect. 4 presents the overall system architecture on which our contributions are based on. Section 5 describes the prototypical implementation while, finally, Sect. 6 and Sect. 7 discuss related work and draw some concluding remarks, respectively.

2 Background and Motivations

We hereafter introduce the fundamental notions and terms needed in the rest of this paper. We also illustrate a simple yet effective example motivating our work.

2.1 Deployment Models and Deployment Technologies

For automating the deployment of an application, *deployment models* are typically used to describe the desired result: In general, there is a distinction between *imperative deployment models* and *declarative deployment models* [15]. Declarative models, in general, declare exactly *what* the desired state into which an application or parts thereof are transferred to. In contrast, imperative models define the exact process of *how* the desired state is reached using executable workflows or programmatic actions. Hence, a declarative deployment model specifies the structure of components to be deployed and defines the desired state in the form of properties or configurations for those components, but it requires a deployment technology that interprets the model and derives the exact order of operations to reach this state. For example, in Terraform an application developer creates a set of files defining the cloud resources the foreseen application requires. Terraform, when executing the application deployment, analyzes the resource definitions and derives a workflow having exact steps and actions required to roll-out the desired state defined by the application developer.

In industry and research, declarative deployment models are widely accepted as the most appropriate approach for application deployment and configuration management [17]. As a result, a plethora of different technologies have been developed following this approach such as Chef, Puppet, AWS CloudFormation, Terraform, and Kubernetes. However, application systems are often in constant change and, besides the major effort for adapting the application itself, the associated deployment models must be adapted using different or additional deployment technologies. Deployment technologies are heterogeneous regarding supported features and modeling languages, and this could result in major efforts whenever an application and its actual deployment have to be adapted to changes or evolutions in the application requirements. Therefore, it is crucial to postpone as late as possible the choice of which deployment technology to use. An even better approach for application developers is to define their application structure and desired state in a technology-agnostic manner, e.g., by exploiting a normalized metamodel. With a normalized modeling of the application and of its desired state, one can indeed automatically generate the deployment artifacts needed to deploy the application using a given deployment technology.

In our previous work [31], a systematic review of widely used declarative deployment technologies revealed the Essential Deployment Metamodel

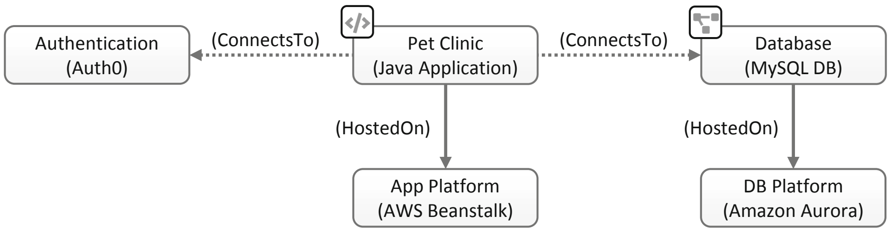


Fig. 1. Simple cloud application which can only be deployed by using a subset of one of the top-most deployment technologies.

(EDMM). EDMM provides a normalized metamodel as a technology-independent baseline for deployment automation research and provides a common understanding of declarative deployment models. EDMM comprises the essential parts supported by well-known technologies and facilitates the transformation in different concrete technologies by a semantic mapping, which avoids deployment technology lock-in.

2.2 Motivating Scenario

As a motivating scenario for our work, we consider a rather simple cloud application. Figure 1 depicts the scenario and shows a Java application, named "Pet Clinic", in the center that is hosted on *AWS Beanstalk*, the platform as a service (PaaS) offering by Amazon Web Services (AWS). This application connects to a fully managed database platform, *Amazon Aurora* which is a managed MySQL database as a service (DBaaS) offering by AWS. Both the Java application as well as the Database component have an artifact attached (cf. Fig. 1), which is, for example, a packaged JAR file in case of the Java application and a SQL file containing the actual database schema and initial data in case of the database component. The left hand side of Fig. 1 depicts a software as a service (SaaS) offering. For this scenario, we envision the usage of a managed authentication service to provide single sign-on between different applications. The Java application, therefore, needs to connect or redirect users to this authentication service to authenticate and authorize them.

Even if simple, this application cannot be deployed by various deployment technologies (and by almost all of the most popular technologies we analysed in our previous work [31]). This scenario, as it is, is only fully supported by Terraform, as Terraform provides different plugins for different cloud providers and services. Indeed, parts of the application structure are supported by other deployment technologies as well. For example, AWS CloudFormation, Ansible, and Chef are capable to deploy applications to AWS Beanstalk. However, SaaS hosted components are not widely supported—Terraform supports many popular SaaS offerings. Alternatively, custom deployment automation tools are required that are most likely offered by SaaS providers.

To fully automate the deployment, a decision support system is needed to determine which declarative deployment technologies can be used to fully deploy a given application deployment model. It is important that application developers receive early deployability feedback immediately while modeling the application. Further, to overcome the technology-specific differences, EDMM as a normalized metamodel provides a solid baseline for deployment automation research and a common understanding of declarative deployment models. The knowledge of essential parts supported by well-known technologies facilitates transformation to different deployment technologies, which avoids deployment technology lock-in.

2.3 Essential Deployment Metamodel

The EDMM was introduced as the result of a systematic review of technologies that contain the essential elements of declarative deployment models to enable the comparison and selection of appropriate technologies [31]. The EDMM enables a common understanding of declarative deployment models and, thus, eases the comparison and selection of appropriate technologies. It defines *Components* as physical, functional, 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. Finally, 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. The terminology of EDMM is considered as baseline in the course of this paper.

3 Transforming EDMM Models into Deployment Technology-Specific Models

In this section, we introduce our approach to transform a technology-independent application deployment model based on EDMM into a *deployment technology-specific model* (DTSM) while ensuring supportability by respective deployment technologies. As depicted in Fig. 2, the approach is structured in four steps: (1) Create EDMM Model, (2) Check Supportability with Different Technologies, (3) Transform EDMM Model into DTSM, and (4) Execute DTSM. In the following, we will provide details on each of such steps. Notably, the grey-dashed boxes represent already existing building blocks [30] that are extended in this work, while dark boxes highlight the new main contributions of this work.

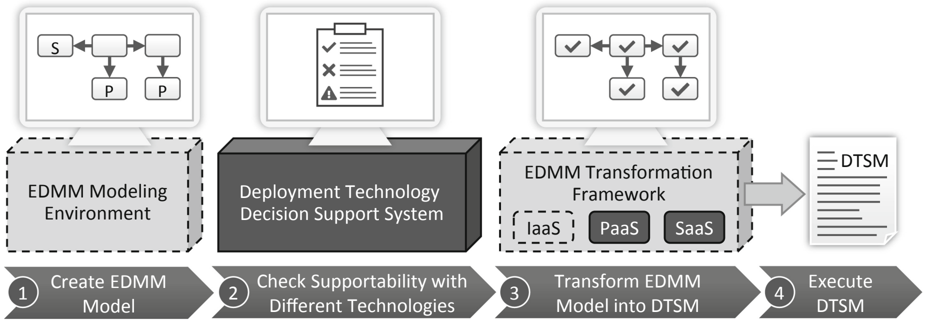

S
P
P
EDMM Modeling Environment
Deployment Technology Decision Support System
EDMM Transformation Framework
IaaS
PaaS
SaaS
DTSM
1 Create EDMM Model
2 Check Supportability with Different Technologies
3 Transform EDMM Model into DTSM
4 Execute DTSM

Fig. 2. Transformation of EDMM Models while ensuring their transformability to specific deployment technologies (based on [30])

3.1 Step 1: Technology-Independent Application Modeling

The modeling of the application is done in EDMM to provide a normalized and technology-independent model. The model is composed graphically by using the *EDMM Modeling Environment* that we proposed in our previous work [30]. The application developer uses the modeling environment to compose a cloud application that, for instance, has the structure as depicted in Fig. 1. The creation of certain EDMM components is based on existing types that are provided by the modeling environment. At any time, the resulting model is compliant to the EDMM in YAML specification[1] and can be exported. To improve the modeling experience and to tackle the issue that an application developer needs live feedback whether a certain deployment technology is capable of deploying the current model, the modeling environment uses the *Deployment Technology Decision Support System*, which is presented next.

3.2 Step 2: Check Deployment Technology Support

In this work, we introduce the *Deployment Technology Decision Support System* building block as shown in Fig. 2. Having this, an application developer can immediately check whether a EDMM model can be transformed into a *deployment technology-specific model* (DTSM) used by a certain deployment technology. The latter obviously holds if the EDMM model includes entities and features supported by a deployment technology. For example, the user gets immediate feedback if a modeled application is supported by Terraform, AWS CloudFormation, Juju, or Ansible, to name just a few. Hereby, the EDMM modeling environment triggers the decision support module whenever an application developer changes the EDMM model. This module consumes the current EDMM model and checks whether and to which degree the model is transformable into a DTSM of a specific deployment technology. The decision support module generates a

[1] https://github.com/UST-EDMM/spec-yaml.

report that is presented to application developer. Based on this report, we facilitate decision support by checking transformability into a specific deployment technology.

3.3 Step 3: Transform EDMM Model into DTSM

For transformation, the EDMM model is consumed by the *EDMM Transformation Framework* module. In this work, we build on top of the existing EDMM Transformation Framework, which we proposed in a previous work [30]. This system is already able to transform EDMM models containing virtual compute resources (IaaS), i. e., operating systems, virtual machines, or containers, and the software that needs to be deployed on them including their configuration and orchestration. To further support cloud application scenarios, we extend the module to comprise certain transformation rules for *PaaS* and *SaaS* component types such that EDMM models containing these can be transformed into respective deployment technology-specific models (DTSM). For example, there are transformation rules for AWS CloudFormation to transform possibly modeled PaaS components. Further, we provide rules, e. g., for Terraform, to transform respective SaaS components into the deployment technology's counterpart. Due to the extensibility and pluggable architecture of the EDMM Transformation Framework, this only leads to changes in the respective plugins to implement the transformation rules accordingly for PaaS and SaaS.

3.4 Step 4: Technology-Specific Deployment Execution

The output of the EDMM Transformation Framework is a deployment technology-specific model (DTSM). For example, in Terraform this will be a configuration that consists of one or more `*.tf` files referencing respective artifacts to deploy. In our approach, we deliberately output technology-specific models to facilitate DevOps activities such as infrastructure as code (IaC) in modern software development environments. By producing human- and machine-readable model files, we enable that transformed results are managed using version control system, e. g., to trigger Git-based continuous integration and delivery (CI/CD) workflows. Notably, by simply re-running the EDMM Transformation Framework targeting a different deployment technology, the same EDMM model can be used to generate the respective technology-specific deployment model [30].

4 System Architecture of the EDMM Modeling, Decision Support, and Transformation System

Figure 3 shows the overall system architecture of the proposed approach. To support the depicted approach from above, several components are required. The *Modeling Tool* is a web-based modeling environment that uses a *REST API* to retrieve and update its data. The *Types Repository* contains reusable EDMM component types that an application developer can use for modeling and provide

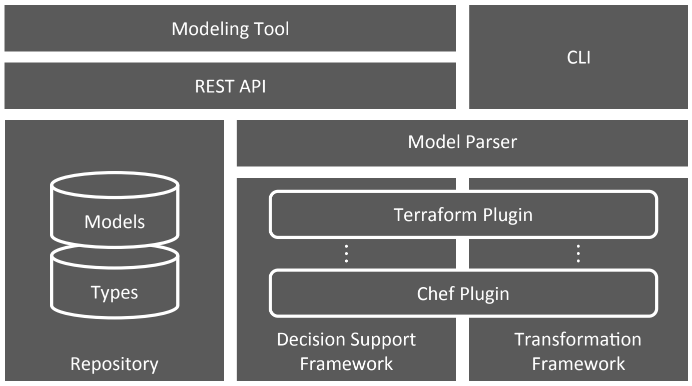


Fig. 3. System architecture supporting modeling, decision support, and transformation of EDMM models into DTSMs.

the respective technical and platform abstractions. An application developer uses these types through the *Modeling Tool* to graphically compose the structure of the EDMM model, which are stored and manged in the *Models Repository*.

To check the transformation support and facilitate decision support, the *Decision Support Framework* is introduced as depicted in Fig. 3. To transform an EDMM model the *Transformation Framework* is envisioned. Both components employ a plugin architecture that supports the integration of various deployment technologies in an extensible and pluggable way. Each plugin employs the knowledge whether a certain EDMM component is supported for transformation or not. The Decision Support Framework is able to utilize the plugins to check a given EDMM model and to produce a report what components (or component types) are not supported. Further, the plugins carry the logic and transformation rules to transform an EDMM model into a deployment technology-specific model (DTSM), which includes the creation of respective technology-specific directory structures, files, and artifacts. The *Model Parser* consumes a textual EDMM model in YAML and creates an internal data structure used by the Decision Support Framework, the Transformation Framework, and the respective plugins.

In addition, the system offers a command-line interface (CLI) that can be either used directly by the user or integrated into any automated workflow, e. g., to facilitate IaC by using it within a CI/CD pipeline. Either way, using the CLI or the web-based interface, an application developer can select the desired target deployment technology in which an EDMM model should be transformed.

5 Validation: Prototypical Implementation

In this section, we illustrate a prototypical implementation of the proposed approach and the foreseen system architecture. As mentioned before, we base our prototype on two existing components: (i) Eclipse Winery [21] as the EDMM Modeling Environment and (ii) the EDMM Transformation Framework [30].

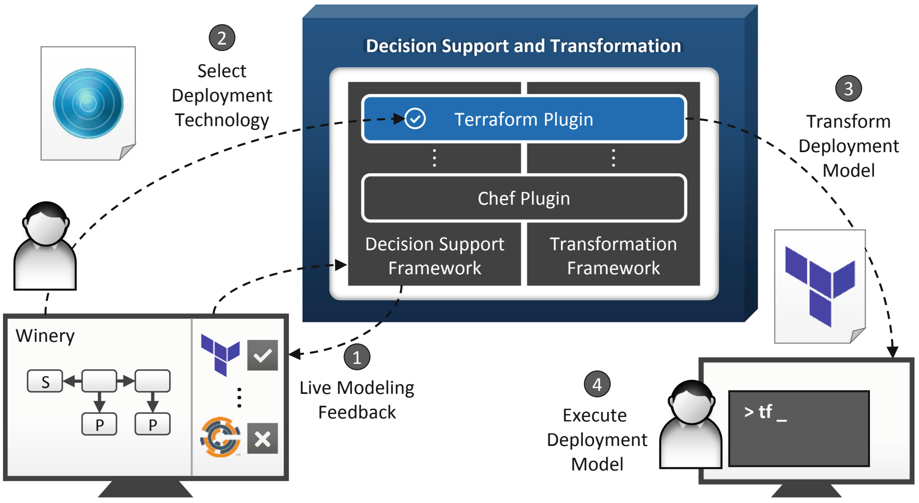
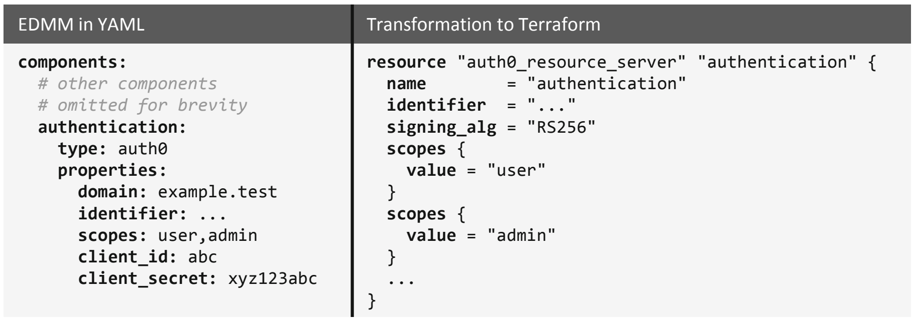


Fig. 4. Prototype flow demonstrating the modeling, decision support, and transformation of an EDMM model to Terraform.eps

5.1 Overview

Eclipse Winery is a web-based environment to graphically model TOSCA-based application topologies. It provides a *back-end* to manage component and relation types, their property definitions, operations, and artifacts. Further, it provides a *Topology Modeler* component which enables the graphical composition of application deployment models including the specification of the components' properties. Even though Winery was initially developed as TOSCA modeling environment, in previous work we showed that EDMM can be mapped to TOSCA [31].

First of all, we extended the EDMM modeling language and introduced new built-in types to respectively cover the motivation scenario depicted in Fig. 1. We extended Winery's Topology Modeler in order to provide a live checking of application models. Winery calls the *Decision Support Framework* whenever the application developer changes the EDMM model, e. g., when adding or removing components. For this purpose, the EDMM Transformation Framework was extended by the *Decision Support Framework* component. Due to the fact that the EDMM Transformation Framework employs a plugin architecture, we extended the existing plugin interface and its `checkModel()` lifecycle method to return a respective result set that highlights the components that are not supported. The communication between Winery and the EDMM Transformation Framework is achieved using REST over HTTP. In addition to the existing CLI of the EDMM Transformation Framework, we now also provide a REST API over HTTP to trigger the transformation for a certain target deployment technology.


EDMM in YAML
Transformation to Terraform

```yaml
components:
  # other components
  # omitted for brevity
  authentication:
    type: auth0
    properties:
      domain: example.test
      identifier: ...
      scopes: user,admin
      client_id: abc
      client_secret: xyz123abc
```

```hcl
resource "auth0_resource_server" "authentication" {
  name        = "authentication"
  identifier  = "..."
  signing_alg = "RS256"
  scopes {
    value = "user"
  }
  scopes {
    value = "admin"
  }
  ...
}
```

Fig. 5. Terraform transformation mapping.

To use the prototype[2], we created a Docker Compose configuration able to start a pre-configured and ready-to-use EDMM Modeling, Decision Support, and Transformation System. All changes and improvements in the course of this paper have been merged to the *master* branches of the respective repositories.

5.2 Modeling and Transformation Flow

In this section, we show the overall modeling, decision support, and transformation flow of our implemented prototype. The flow is explained based on a modeling example that follows our motivating scenario in Fig. 1. Further, we chose Terraform to describe the flow based on a concrete deployment technology.

Application developers start the integrated EDMM Modeling, Decision Support, and Transformation System. By using the EDMM Modeling Tool, users are able to model their desired application structure. As depicted in Fig. 4, the user composes the structure by drag-and-drop desired components to the canvas. Additionally, users define respective relations between them by connecting the components. To facilitate decision support, we implemented live modeling feedback directly in the modeling environment (cf. 1 in Fig. 4). Whenever, the overall model is changed, the EDMM Decision Support Framework is triggered. All available plugins of the Decision Support and Transformation Framework are queried to check if the current model contains unsupported components. The modeling environments retrieves the result and presents it to the application developer. For example, a model that reflects the scenario depicted in Fig. 4, can be transformed into "Terraform" but not into "Chef". If a model is supported by one or more deployment technologies, application developer can export the model according to the EDMM in YAML specification. From here, the user executes the transformation, i. e., using the EDMM CLI, and selects the desired and supported deployment technology (cf. 2 in Fig. 4). The output of the system is the transformed output according to the need a corresponding deployment technology requires. For example, in case of Terraform, it will be a

[2] https://github.com/UST-EDMM.

ready to use working directory containing Terraform configuration files (cf. 3 in Fig. 4). Lastly, the application developer is able to execute the actual deployment using the tools and interfaces provided by the deployment technology. For instance, Terraform provides a CLI to "apply" the generated configuration. At this point, application developers can use their well-known development environments and tools to deploy and manage their applications (cf. 4 in Fig. 4). For example, the generated deployment artifacts can be versioned in revision control systems, such as Git, to facilitate the use of automated CI/CD pipelines.

We executed the modeling and transformation flow according to our motivation scenario from Sect. 2.2 (the full EDMM modeling example in YAML is available online[3] on GitHub). In Fig. 5, we show an excerpt a modeled EDMM-based SaaS component and its mapping to the actual Terraform resource. In such cases, the system generates a respective `auth0_resource_server` resource that maps to the corresponding properties. For this special case, the Terraform plugin comprises a special transformation rule to split the comma-separated list of the EDMM property `scopes` into separate `scopes` blocks.

6 Related Work

The problem of automating the deployment of multi-component applications on cloud platform is well-known [29], with most of existing approaches being declarative [6]. The OASIS standard TOSCA [23,24] is one of the most known approaches in this direction, as it provides a standardized language for specifying multi-component application in a portable way. Specified applications can then be deployed on cloud infrastructures, provided that the latter support the declarative processing of TOSCA application specifications, e. g., by featuring the OpenTOSCA runtime [7]. Our approach differs from TOSCA, as we aim at automatically generating the deployment artifacts needed to deploy an application with an existing technology as it is.

Similar considerations apply to other approach à la TOSCA, e. g., CAMEL [1], MODAClouds [12], Panarello et al. [26], SeaClouds [8] and transcloud [9], just to mention some. Starting from a vendor-agnostic specification of a multi-component application, all such approaches enable its deployment on heterogeneous clouds. This is done by relying on additional components offered by the targeted clouds or on ad-hoc middleware platforms processing the application specification to deploy their components on heterogeneous clouds. Our approach also starts from agnostic representations of multi-component applications, but it rather automatically generates different deployment artifacts for different deployment technologies, in order to directly utilize them to deploy applications on heterogeneous clouds.

In this perspective, closer approaches to ours are those by Di Cosmo et al. [10,11] and by Guillén et al. [16], which both share our baseline idea of generating concrete deployment artifacts from a vendor-agnostic specification of a multi-component application and of its desired configuration. Di Cosmo et al.

[3] http://bit.ly/3akWSYR.

indeed propose a solution for automatically synthesizing a concrete deployment for a multi-component application in a cloud environment, based on a high-level specification of the application and its desired state. Their solution is however targeting OpenStack cloud deployments, while we target 13 different production-ready deployment technologies, each allowing to deploy applications on various different cloud infrastructures [30].

Guillén et al. [16] instead present a framework for developing multi-service application that are decoupled from the architecture, services, and libraries provided by cloud vendors. Based on additional metadata indicating application requirements, the framework generates cloud compliant software artifacts that are deployed in each cloud platform. This approach is even closer to ours, as the same application can be deployed differently by re-running the framework and instructing it to target different clouds. The approach by Guillén et al. however differs from ours since it is intended to process applications whose sources are available to the framework, while our approach only considers the application specification and the final packaged software artifact. This allows us to process a wider set of applications, as we allow developers to reuse *black-box* third-party software or SaaS services to implement the components of an applications. Similar considerations apply to the solution proposed by Alipour and Liu [3], who exploit model-to-model transformation to obtain a cloud specific application deployment from a vendor-independent application.

Other solutions worth mentioning are OAM [22], Kompose [28] and Compose Object [13]. The OAM has recently been proposed to allow developers and operators to separately describe containerized applications with a vendor-agnostic representation. It indeed allows developers to describe what containerized components do and how they should be configured, while operators can complete application specifications by configuring runtime environments. Obtained application specifications can then be run on Kubernetes with Rudr[4]. Our approach can be used for the same purposes, and it can be used not only for running containerized applications on Kubernetes, but also for running other types of applications on other deployment technologies. Similar considerations apply to Kompose and Compose Object, both enabling the deployment of containerized applications on Kubernetes. Kompose does so by automatically generating a Kubernetes deployment for containerized applications specified in Docker Compose, while Compose Object is a Kubernetes plugin for directly running such a kind of applications on Kubernetes clusters.

It is worth noting that our approach of transforming EDMM models to deployment artifacts is essentially a M2M (Model-to-Model) transformation [20]. We could have hence implemented our approach by suitably configuring existing frameworks, e. g., ATL [19], QVTd [14], or ADOxx [2], which already come with tooling for graphical modeling and transformation. However, we decided to implement our solution as a lightweight command-line tool, as it offers a convenient way to be integrated in CI/CD pipelines and supporting DevOps [5].

[4] https://github.com/oam-dev/rudr.

It is also worth noting that our approach is inspired by the work by Papazoglou and van den Heuvel [27], who firstly outlined the possibility of *blueprinting* cloud-based application deployments, i. e., specifying the deployment of multi-component applications in a reusable way, and to exploit such specifications to automate application deployments. Such a foundational idea is the rationale behind our EDMM modeling and transformation framework. Our framework was also inspired by Andrikopoulos et al. [4], who firstly investigated the commonalities among existing cloud modeling languages and collected them in the so-called GENTL topology language. In our previous work [31] we followed a similar approach for obtaining the EDMM itself, which we then exploit in this and former work to develop the EDMM modeling and transformation framework.

In summary, to the best of our knowledge, ours is the first approach automatically generating the artifacts needed to process multi-component applications using different existing deployment technologies by also allowing to reuse third-party software and SaaS services to implement some components of an application. It does so by starting from the widely accepted idea of specifying an application in a technology-agnostic way, without requiring cloud providers to support additional runtimes, and by piggybacking on existing, production-ready deployment technology to actually enact application deployments.

7 Conclusions and Future Work

The EDMM modeling and transformation framework [30,31] allows to deploy a multi-component application using different, existing declarative deployment technologies. It indeed features a YAML-based language distilling the essentials of existing technologies, which allows to describe a multi-component application and its desired state. Deploying an application or migrating from a deployment to another then only requires to feed the EDMM transformation framework with the application specification. By selecting the target deployment technology, the transformation framework will automatically generate the deployment artifacts needed to deploy the specified application using such technology. This currently comes at the price of only exploiting IaaS-based virtual machines or containers as compute nodes where to deploy the components of an application.

In this paper, we presented an extension of the EDMM Modeling and Transformation Framework allowing to deploy application components on PaaS platforms and to exploit existing SaaS services to implement components of an application. We also proposed a decision support system allowing to determine which declarative deployment technologies can actually be used to deploy an application specified with EDMM, as some existing technology may not be offering all features needed to deploy the specified application (e. g., Juju and CFEngine are not supporting the deployment application components on PaaS platforms). To illustrate the helpfulness of our extension, we also shown how it was exploited on a running example, which, despite simple, would have not be addressed by the original EDMM Modeling and Transformation Framework.

The contributions in this paper present a first step towards cloud-native application deployments using EDMM. However, in future work, it needs to be analyzed which general features a declarative deployment technology has to support to deploy *arbitrary* cloud-native applications comprising, e. g., FaaS components and arbitrary other managed services such as message queues. Therefore, we will first analyze the requirements on deployment technologies to support deploying arbitrary cloud-native applications and integrate required mechanisms and plugins afterwards into our system. Further, we plan to extend the intelligence of the decision support system by allowing to measure the *distance* from an application specification to its deployment on a given technology, e. g., in terms of the least amount of adaptation updates that must be applied to the application to allow its deployment on such technology. We also plan to include an adaptation recommender in the decision support system, capable of indicating to the application developer the changes to apply to an application to allow its deployment on a desired technology, e. g., indicating to replace the PaaS platform used to host some components with a IaaS-based software stack, so as to enable the deployment of an application on Juju and CFEngine.

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

References

1. Achilleos, A.P., et al.: The cloud application modelling and execution language. J. Cloud Comput. **8**(1), 1–25 (2019). https://doi.org/10.1186/s13677-019-0138-7
2. ADOxx: ADOxx.org (2020). https://www.adoxx.org. Accessed 13 Feb 2020
3. Alipour, H., Liu, Y.: Model driven deployment of auto-scaling services on multiple clouds. In: 2018 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 93–96 (2018)
4. Andrikopoulos, V., Reuter, A., Gómez Sáez, S., Leymann, F.: A GENTL approach for cloud application topologies. In: Villari, M., Zimmermann, W., Lau, K.-K. (eds.) ESOCC 2014. LNCS, vol. 8745, pp. 148–159. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44879-3_11
5. Belmont, J.M.: Hands-On Continuous Integration and Delivery, 1st edn. Packt Publishing, Birmingham (2018)
6. Bergmayr, A., et al.: A systematic review of cloud modeling languages. ACM Comput. Surv. **51**(1), 1–38 (2018)
7. Binz, T., et al.: OpenTOSCA – a runtime for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 692–695. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_62
8. Brogi, A., et al.: EU project seaclouds - adaptive management of service-based applications across multiple clouds. In: Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014), pp. 758–763. SciTePress (2014)
9. Carrasco, J., Durán, F., Pimentel, E.: Trans-cloud: CAMP/TOSCA-based bidimensional cross-cloud. Comput. Stand. Interfaces **58**, 167–179 (2018)

10. Di Cosmo, R., Eiche, A., Mauro, J., Zacchiroli, S., Zavattaro, G., Zwolakowski, J.: Automatic deployment of services in the cloud with Aeolus Blender. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 397–411. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_28
11. Di Cosmo, R., et al.: Automated synthesis and deployment of cloud applications. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, pp. 211–222. ACM (2014)
12. Di Nitto, E., Matthews, P., Petcu, D., Solberg, A. (eds.): Model-Driven Development and Operation of Multi-Cloud Applications: The MODAClouds Approach. SAST. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-46031-4
13. Docker Inc: Compose Object (2020). https://github.com/docker/compose-on-kubernetes. Accessed 13 Feb 2020
14. Eclipse Foundation: Eclipse QVTd (QVT Declarative) (2020). https://projects.eclipse.org/projects/modeling.mmt.qvtd. Accessed 13 Feb 2020
15. Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., Wettinger, J.: 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), pp. 22–27. Xpert Publishing Services (2017)
16. Guillén, J., Miranda, J., Murillo, J.M., Canal, C.: A service-oriented framework for developing cross cloud migratable software. J. Syst. Softw. **86**(9), 2294–2308 (2013)
17. Herry, H., Anderson, P., Wickler, G.: Automated planning for configuration changes. In: Proceedings of the 25th International Conference on Large Installation System Administration (LISA 2011), pp. 57–68. USENIX (2011)
18. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Addison-Wesley, Boston (2010)
19. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: a model transformation tool. Sci. Comput. Program. **72**(1), 31–39 (2008)
20. Kahani, N., Bagherzadeh, M., Cordy, J.R., Dingel, J., Varró, D.: Survey and classification of model transformation tools. Softw. Syst. Model. **18**(4), 2361–2397 (2018). https://doi.org/10.1007/s10270-018-0665-6
21. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – a modeling tool for TOSCA-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 700–704. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_64
22. Microsoft and Alibaba Cloud: Open Application Model (2020). https://oam.dev. Accessed 13 Feb 2020
23. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013)
24. OASIS: TOSCA Simple Profile in YAML Version 1.2. Organization for the Advancement of Structured Information Standards (OASIS) (2019)
25. Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do internet services fail, and what can be done about it? In: Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems (USITS 2003). USENIX (2003)

26. Panarello, A., Breitenbücher, U., Leymann, F., Puliafito, A., Zimmermann, M.: Automating the deployment of multi-cloud applications in federated cloud environments. In: Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools, pp. 194–201. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST) (2017)
27. Papazoglou, M.P., van den Heuvel, W.J.: Blueprinting the Cloud. IEEE Internet Comput. **15**(6), 74–79 (2011)
28. The Kubenetes Authors: Kompose (2020). https://kompose.io. Accessed 13 Feb 2020
29. Wettinger, J., Andrikopoulos, V., Leymann, F., Strauch, S.: Middleware-oriented deployment automation for cloud applications. IEEE Trans. Cloud Comput. **6**(4), 1054–1066 (2018)
30. Wurster, M., et al.: The EDMM modeling and transformation system. In: Service-Oriented Computing – ICSOC 2019 Workshops. Springer, December 2019
31. Wurster, M., et al.: The essential deployment metamodel: a systematic review of deployment automation technologies. SICS Softw.-Intensive Cyber-Phys. Syst. (2019). https://doi.org/10.1007/s00450-019-00412-x