Automating the Deployment of Distributed Applications by Combining Multiple Deployment Technologies

Michael Wurster[1], Uwe Breitenbücher[1], Antonio Brogi[2], Felix Diez[1],
Frank Leymann[1], Jacopo Soldani[2] and Karoline Wild[1]
[1] *University of Stuttgart, Institute of Architecture of Application Systems, Stuttgart, Germany*
[2] *University of Pisa, Department of Computer Science, Pisa, Italy*
[lastname]@iaas.uni-stuttgart.de, [lastname]@di.unipi.it

Keywords: Distributed Application Deployment, Automation, EDMM, EDMM Framework, Cloud Computing.

Abstract: Various deployment technologies have been released to support automating the deployment of distributed applications. Although many of these technologies provide general-purpose functionalities to deploy applications as well as infrastructure components, different technologies provide specific capabilities making them suited for different environments and application types. As a result, the deployment of complex distributed applications often requires to combine several deployment technologies expressed by different deployment models. Thus, multiple deployment models are processed by different technologies and must be either orchestrated manually or the automated orchestration must be developed individually. To address these challenges, we present an approach (i) to annotate parts of a holistic deployment model that should be deployed with different deployment technologies, (ii) to automatically transform an annotated model to multiple technology-specific models for different technologies, and (iii) to automatically coordinate the deployment execution with different technologies by employing a centralized orchestrator component. To prove the practical feasibility of the approach, we describe a case study based on a third-party application.

1 INTRODUCTION

Automating the deployment of complex distributed applications is crucial nowadays, as it enables fully exploiting the potentials of cloud computing. At the same time, manually executing the deployment of complex systems is error-prone, time-consuming, and costly (Oppenheimer et al., 2003; Brogi et al., 2018).

Various technologies, e. g., Terraform, CloudFormation, or Ansible, and standards, e. g., the Topology and Orchestration Specification for Cloud Applications (TOSCA) (OASIS, 2015), have been released to accomplish the need for deployment automation. Most of them follow a *declarative* approach, which is considered the de-facto standard for application deployment in industry and research (Wurster et al., 2019b). They indeed all feature a Domain-Specific Language (DSL) for specifying the desired state for an application by means of a structural description of the application components, their relations, and configuration. Deployment automation is then achieved by processing application specifications to automatically derive the operations to be executed in the exact order to reach the desired state (Endres et al., 2017).

However, the deployment of complex distributed applications often requires to combine different deployment automation technologies (Di Nitto et al., 2017; Guerriero et al., 2019). Many deployment automation technologies offer multi-purpose functionalities for deploying different types of applications on different infrastructure components or cloud services. Each technology however provides specific capabilities, e. g., Terraform focuses primarily on cloud infrastructure and cloud service provisioning for different cloud platforms, while CloudFormation is tailored for managing only AWS resources. Moreover, technologies such as Chef or Ansible are instead specialized in the configuration management of software components on running virtual machines (VMs). Thus, the combination of different deployment automation technologies enables to concretely enact application deployments, e. g., Terraform to provision a VM on a cloud platform, plus Ansible to manage arbitrary software components on top of it. This is because there is no "one fits all" deployment automation technology, i. e., no existing technology supports deploying arbitrary application components on arbitrary computing environments or platforms.

As a result, application developers are required to write multiple deployment models to deploy their applications, with each model describing a part of the overall application deployment with a different DSL. This requires the appropriate expertise in different technologies and changes of the overall application structure may require the adaptation of multiple deployment models. In addition, the processing of diverse deployment models requires to orchestrate the respective deployment technologies in the right order, i. e., to invoke respective APIs, which have to be coordinated manually. Currently, no existing approach supports a single deployment model whose parts can be processed by different deployment technologies in an automated manner. Thus, the overall research question is *"How to seamlessly model and automate the deployment of a complex application distributed across heterogeneous environments that requires different deployment technologies?"*

The contributions of this paper are twofold. First, it presents an approach based on a holistic deployment model that expresses the deployment of the overall application and enables (i) to annotate parts that must be deployed by different technologies, (ii) to automatically translate these parts into *deployment technology-specific models* (DTSMs) that can be processed by the respective technologies, and (iii) to automatically coordinate the overall deployment with all target technologies. For this purpose we have built on existing work and use the *Essential Deployment Metamodel* (EDMM) (Wurster et al., 2019b), which describes the essential modeling entities supported by the majority of deployment technologies, and shows how such a model can be partitioned and the execution orchestrated. So far only the use of one technology has been possible. The second contribution is an extended system architecture of the EDMM Framework (Wurster et al., 2019a, 2020a) (i) to transform an EDMM model in multiple DTSMs considering the inter-dependencies between them and (ii) to automate application deployments by orchestrating multiple deployment technologies. Hereby, the deployment is enacted by only relying on APIs of a deployment technology, while ensuring that the deployment information are exchanged adequately between them. A prototypical implementation of the system architecture and a case study using Kubernetes, Terraform, and Ansible demonstrate the overall practical feasibility.

In the following, Sect. 2 presents fundamentals and Sect. 3 motivates our work. Sect. 4 introduces our approach and Sect. 5 proposes the system architecture. Sect. 6 describes the prototype and Sect. 7 discusses the contributions while Sect. 8 and Sect. 9 discuss related work and draw some concluding remarks.

2 FUNDAMENTALS

This section presents the fundamentals of deployment automation and introduces EDMM.

2.1 Deployment Models & Deployment Technologies

In practice, deployment technologies typically use *declarative deployment models* to describe the desired outcome of an automated deployment of an application (Herry et al., 2011; Wurster et al., 2019b; Bergmayr et al., 2018). Many deployment automation technologies have been developed and many of them have originated from the industry, such as Chef, Puppet, AWS CloudFormation, Terraform, or Kubernetes. Each deployment automation technology uses its own DSL to declaratively model application deployments, leading to a vendor lock-in with respect to the used deployment model and technology. Further, there is no "one fits all" deployment technology that can be used for arbitrary use cases and many of them are tailored for certain use cases and only offer capabilities to target certain infrastructure components or cloud services. Even if such technologies share the same purpose, they differ in features and supported mechanisms. For example, someone would use Terraform to provision cloud infrastructure and cloud services such as VMs or managed database offerings from multiple vendors while using configuration management tools such as Chef or Ansible to install and configure arbitrary software components on these VMs that utilize the provisioned cloud services.

Existing model-driven approaches that enable the deployment of distributed applications in multi-cloud environments or across organizational boundaries mainly focus on the automated refinement of provider-independent models to provider-specific models for respective hosting environments (Di Nitto et al., 2017) or the distributed deployment and management (Sebrechts et al., 2018; Saatkamp et al., 2019; Wild et al., 2020). However, the combination of different *deployment automation technologies* is not covered yet and requires still manual modeling, transformation, and coordination effort.

2.2 Essential Deployment Metamodel

The essential modeling entities supported by the majority of declarative deployment technologies have been extracted by investigating the 13 most used deployment technologies (Wurster et al., 2019b). The Essential Deployment Metamodel (EDMM) enables a common understanding of declarative deployment

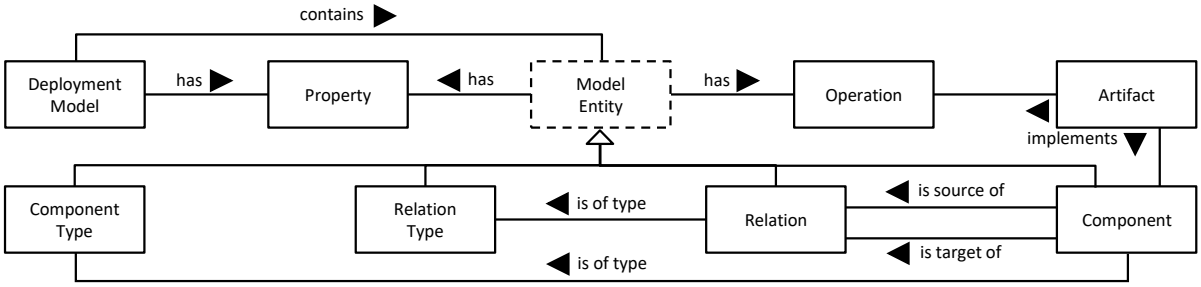

Figure 1: Essential Deployment Metamodel (EDMM) [adapted from Wurster et al. (2019b)].

models and eases the comparison and selection of appropriate technologies. Further, it facilitates the transformation of a model employing the EDMM modeling entities into the DSL of one of the 13 most used deployment technologies (Wurster et al., 2019a). We now recap the EDMM entities and terminology required to understand this paper[1].

Figure 1 depicts the essential modeling entities of EDMM. *Components* enable modeling physical, functional, or logical units of an application. For example, a deployment model may contain several components representing the deployment of a Java web application, a Tomcat web server, or the provisioning of an Ubuntu virtual machine (VM). Further, *Component Types* define reusable entities that specify the semantics of a component that have this type assigned. What required to install or terminate a component are provided by its type in the form of *Properties* and *Operations*. *Properties* describe the desired target state or configuration for a component. *Operations* instead define executable procedures for managing a component during application deployment. For example, a "Tomcat Web Server" component type may define a "Port" property as a means for configuration as well as a "install" and "start" operation to encapsulate the logic how to install and start it. *Relations* instead enable representing directed physical, functional, or logical dependencies between components. *Relation Types* are reusable entities that define a specific kind of dependency. In particular, relation Types enable distinguishing relations modeling that a component "connects to" to another, e. g., a web application component connecting to a database component, from those modeling that a component is "hosted on" another, e. g., a Tomcat web server installed in a VM.

It is finally worth noting that components represent a certain functionality for a specific application and relations are modeled between exactly two of its components. Component types and relation types can instead be reused in different models.

[1]A more detailed, self-contained presentation of EDMM has been published by Wurster et al. (2019b).

3 MOTIVATING SCENARIO & RESEARCH QUESTIONS

The reasons for using different technologies are manifold. Different deployment technologies only support certain target environments. For example, for provisioning and managing AWS resources CloudFormation fits best. Further, different technologies have different capabilities and purposes. While technologies such as Terraform are focusing on infrastructure management for multiple cloud platforms, the main purpose of configuration management technologies such as Chef are to configure applications on running infrastructure. However, defining, managing, and orchestrating multiple independent deployment models for different parts of an application is costly, requires expertise in each technology, and is error-prone due to manual tasks coordinating the execution.

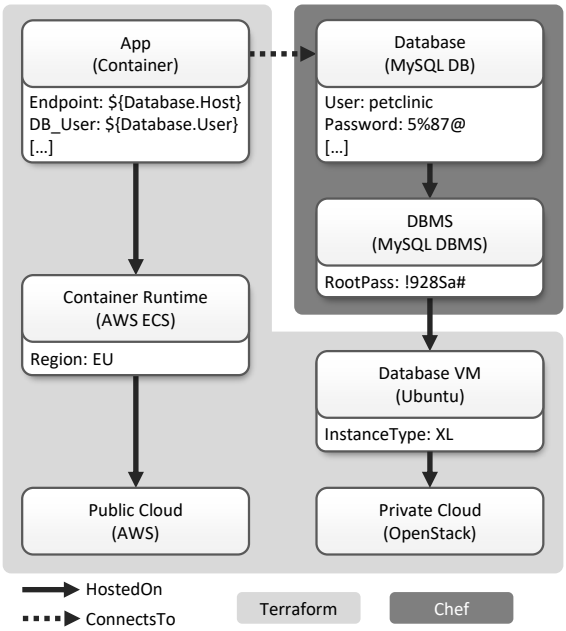

Figure 2: Multi-component application deployment scenario where different parts are defined to be deployed using different deployment automation technologies.

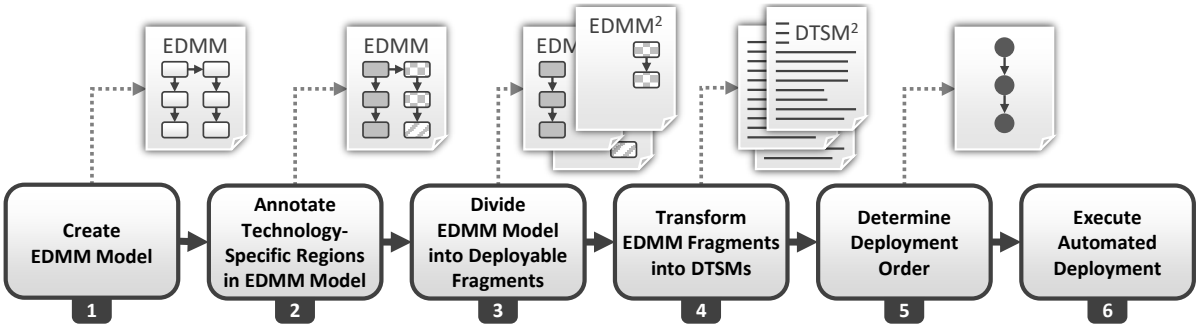

Figure 3: Automating the deployment of distributed applications by combining multiple deployment technologies.

Therefore, we strive to model the complete application system in a *holistic* deployment model. Fig. 2 depicts a simplified motivating scenario which nevertheless reflects practical relevance. The scenario shows a containerized application that is hosted on Amazon's Elastic Container Service (ECS) on the left. The container exposes a web application connecting to a "MySQL" database to store its data. Due to privacy concerns, the database management system is installed on an Ubuntu VM hosted inside an on-premise OpenStack environment. In this scenario, the decision is to use "Terraform" for cloud infrastructure and cloud service provisioning as it can be used with multiple cloud providers and platforms. Further, the configuration management tool "Chef" is used to install and configure arbitrary software components, e. g., the depicted MySQL database management system and its configuration, on top of running infrastructure. Thus, the first research question (RQ) is:

> **RQ 1.** *"How can a holistic deployment model be annotated and divided so that different parts are deployed by different deployment technologies?"*

The motivation scenario depicts that the database components are deployed using a different technology than the container and infrastructure components. This requires to divide and transform the user-defined deployment technology regions, i. e., annotated components, into executable DTSMs. As a result, this requires orchestrating the execution of different models using different deployment technologies. Thus, the second RQ tackled by this paper is:

> **RQ 2.** *"How can the deployment execution be suitably coordinated using multiple deployment technologies, if these rely on different DSLs and provide different APIs?"*

To tackle these research questions in the next section our approach is presented. The remaining details of Fig. 2 are explained in Sect. 4.1.

4 APPROACH

This section introduces an approach to automate the deployment of distributed applications by combining multiple deployment technologies. The overall application is modeled using EDMM and the components are *annotated* with the deployment technology to be used for the actual deployment. Afterwards, the holistic model is *divided* into valid EDMM model fragments that are *transformed* into executable DTSMs. Finally, the execution of all target technologies are coordinated to suitably automate the deployment by employing a plugin-based orchestrator.

Our approach is structured in six steps (Fig. 3): (1) Create EDMM model for the whole application, (2) annotate technology-specific regions in the EDMM model, (3) divide the EDMM model into multiple EDMM model fragments, (4) transform the EDMM model fragments into DTSMs, (5) determine the deployment order of the generated DTSMs, and (6) execute the automated deployment. Steps 1 to 4 address RQ1, while steps 5 and 6 address RQ2.

4.1 Step 1: Technology-Independent Application Modeling using EDMM

The modeling of the application is done using EDMM to provide a normalized and technology-independent model. The application modeling can be graphically edited by exploiting the *EDMM Modeling Tool* as proposed by Wurster et al. (2019a). Application components can be modeled by instantiating an existing component type that is provided by the modeling environment. For example, the "Database" component in our motivating scenario (Fig. 2) is modeled by instantiating the "MySQL DB" component type. Apart from the component name, "Database" in this case, one can define several properties according to the component type's specification to configure the deployment. Similarly, relations can be modeled by connecting two components, either to specify that the

source component "connects to" the target component or that it is "hosted on" the other.

Certain components may require information from components they relate to get successfully deployed. While this could be straightforward when deploying components with the same technology, it may be difficult when components are deployed with different technologies. This requires to suitably coordinate such technologies, especially concerning the information that is only available at runtime. Consider, for example, "App" in Fig. 2, which requires the endpoint and credentials to connect to the MySQL "Database". While the username and password can be directly accessed in the model, the IP address is only available after the VM of the database has been deployed. Therefore, it must be possible to reference certain runtime information of specific components, like in the case of the backend endpoint for "App", which must reference the hostname or IP address of the database VM. Such runtime property references must be provided as inputs to the respective deployment technology prior to the deployment execution.

4.2 Step 2: Annotate Technology-Specific Regions

In this step, the holistically created EDMM model will be annotated to create technology-specific regions indicating which deployment technology should be used for the deployment. According to model-driven architecture (MDA), the components in the model are *marked* with the deployment technology to be used for the actual deployment, i. e., the marked components form a region. For this, the EDMM model syntax has been extended by a new element, called `technology_regions`, which we introduce in the course of this work. This allows to define a map that assigns each component of a deployment model to exactly one deployment technology.

Our motivating scenario in Fig. 2 already shows the assignment of the application components with different groups of components assigned to different deployment technologies: "Database" and "DBMS" are to be deployed using Chef, while the other components are to be deployed using Terraform. Notably, technology regions are not limited to components which are directly connected, as shown by Fig. 2.

4.3 Step 3: Divide EDMM Model into Deployable Fragments

Before DTSMs can be generated, the overall EDMM model has to be divided into deployable EDMM model fragments. A deployable fragment is a group

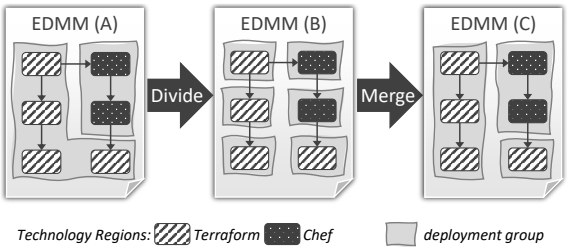

Figure 4: Abstracted EDMM model with assigned technology regions: (A) non-deployable deployment group assignment and the algorithms procedure for (B) dividing and (C) merging deployment groups.

of components in the EDMM model that can be deployed by a certain technology as a *"one-shot deployment"*, i. e., deployed by a single run of the underlying deployment technology. Thus, we call this group of components *deployment group*. The defined technology regions from step 2 are the basis, since all components assigned to a technology region have to be deployed by the same technology.

At the same time, it may be that the components in a technology region cannot be deployed all together. Consider, for instance, the situation in Fig. 4, which shows the application in our motivating scenario by abstracting from component details. EDMM (A) specifies two deployment groups, each containing all components that have to be deployed with the same technology. Two relations have to be realized between components assigned to different groups. This results in a cycle of dependencies between deployment groups, which makes it not possible to automatically process the deployment groups: We cannot first deploy the components in the Terraform group as we first need to run Chef to deploy the components in its group, but a similar argument prevents first deploying the components in the Chef group.

To enable deploying an application like that in EDMM (A) as depicted in Fig. 4, deployment groups have to be refined so that no cycles occur among them. We automatically enact such a refinement by exploiting the *division approach* introduced in previous work as a baseline (Saatkamp et al., 2019). As sketched in Fig. 4, first all deployment groups are partitioned into singleton deployment groups. The obtained groups are then iteratively merged until no more groups can be merged without resulting in cyclic dependencies. The objective is to minimize the number of groups to reduce the coordination effort for collecting and distributing information between the deployment technologies, since for each deployment group, i. e., the resulting DTSM, the required information must be provided as input before it can be executed.

We extended the algorithm presented by Saatkamp et al. (2019) to determine deployment

groups not only based on horizontal relations, e. g., "ConnectsTo" relations, but also vertical relations, e. g., "HostedOn" relations, while considering the technology regions. The rule for merging deployment groups is as follows: Two components can be deployed together if (i) they have the same technology assigned and (ii) merging the components preserves the acyclicity between deployment groups.

4.4 Step 4: Transform EDMM Model Fragments into DTSMs

In this step, each model fragment determined by the last step is transformed into its respective DTSM. We extended the existing EDMM Transformation Framework (Wurster et al., 2019a) to transform an EDMM model fragment into a DTSM of a certain deployment technology. Notably, we achieve this primarily because each determined EDMM model fragment is assigned to a single deployment technology.

The EDMM Transformation Framework enables transforming a given EDMM model into required artifacts, i. e., files and models, to execute the deployment using a selected deployment technology. It is plugin-based and supports the transformation into DTSMs of 13 deployment automation technologies, such as Kubernetes, Terraform, Chef, Ansible, and AWS CloudFormation. Thus, the transformation can be executed for each EDMM model fragment. However, the transformation process needs to be aware of dependencies, i. e., relations, between deployment groups. For example, in a horizontal split as depicted on the right hand side in Fig. 2 between Chef and Terraform, the "IP address" of the provisioned Ubuntu VM must be made available before Chef can be executed. Further, considering the vertical split between Terraform and Chef, the database credentials of the MySQL database need to be made available to the container application prior to deployment. Therefore, we distinguish between configuration parameters, such as port numbers or credentials, that are available through the overall EDMM model, and runtime information that is only available after the deployment of a component, e. g., IP addresses. Depending on the combination of deployment technologies, certain information must be provided from components deployed with one technology to components deployed with another technology. Therefore, the `$`-notation is used for accessing such information, as shown in Fig. 2. We extended EDMM Transformation Framework to (i) resolve configuration parameters using the overall EDMM model and (ii) to translate the usage of runtime properties into respective *inputs* and *outputs* of the underlying deployment technology.

4.5 Step 5: Determine Deployment Order

The *deployment order* is the basis to execute the overall deployment of the generated DTSMs. Thereby, it considers the deployment dependencies between deployment groups since these determine the order how the overall application must be deployed. Further, the deployment order also provides the knowledge which *inputs* and *outputs* for each DTSM execution need to be provided during the automated deployment.

In general, the provisioning dependencies between components is determined by reversing relations, i. e., the target component of a relation must be deployed before the source component (Breitenbücher et al., 2014), which represents the *deployment order* of the connected components. Further, following the definition by Saatkamp et al. (2019), the *deployment order* of the determined deployment groups is calculated by reversing the relations between components in different deployment groups. For the example in Fig. 2, the algorithm would determine three inter-connected deployment groups: two separate groups for Terraform and a third one for Chef, as indicated by (C) the resulting deployment groups shown in Fig. 4. By reversing the relations between groups, and by topologically sorting the resulting graph, the defined order is the following: First execute the Terraform model for the OpenStack infrastructure, then the Chef model to install and configure the MySQL database, and finally the second Terraform model to configure the cloud services in AWS and deploy the container application.

4.6 Step 6: Execute Automated Deployment

Deployment technologies usually provide multiple options to interact with, e. g., by providing software development kits (SDKs), command-line interfaces (CLIs), or APIs such as REST APIs over HTTP. Therefore, suitable *Orchestration Plugins* need to generalize the execution of one of the deployment technology's interaction mechanism. Further, these Orchestration Plugins must handle the required *inputs* and *outputs*, i. e., *collect* information prior to the execution and *distribute* information after the execution. For example, considering the motivating scenario, the plugin for Chef requires information about the target compute environment, i. e., the IP address and SSH credentials, to actually execute the deployment of the MySQL database. Therefore, the Orchestration Plugin for Terraform must populate these information based on the outputs defined by the DTSM.

5 SYSTEM ARCHITECTURE

This section introduces the system architecture to automate application deployments by orchestrating multiple deployment technologies. Fig. 5 shows a component diagram to depict the modular structure of the system, which is an extension of the system architecture of the existing EDMM Modeling and Transformation System (Wurster et al., 2020a). Light gray components represent new components while the shaded component represents an existing one that has been extended in the course of this work.

The *Modeling Tool* (Wurster et al., 2019a) is a web-based modeling environment that uses a *REST API* to retrieve and update its data. All data, e. g., created models or the reusable EDMM component types, is accessible through the REST API and stored in the *Repository*. Users graphically compose the structure of the overall EDMM model using the reusable EDMM component types. The technology-specific regions are annotated by the user after exporting the EDMM model from the Modeling Tool, i. e., covering the first two steps of our proposed approach.

The remaining steps are covered by a CLI that can be used to *divide* and *transform* the overall EDMM model into multiple DTSMs and to *execute* the automated deployment. Therefore, the *Model Parser* and *Model Divider* components are the core components. The Model Parser parses the overall EDMM model and translates it into an internal graph-based data structure on which the Model Divider performs the extended *division approach*.

To transform the EDMM model fragments into DTSMs, we employ the EDMM Transformation Framework (Wurster et al., 2019a) as baseline for the depicted *Transformation* component. It uses a plugin architecture that supports the integration of various deployment technologies in an extensible way. Each plugin encompasses the knowledge whether a certain EDMM component is supported for transformation or not. Further, the plugins carry the logic and transformation rules to transform an EDMM model into a DTSM, which includes the creation of respective technology-specific directory structures, files, and artifacts based on the respective DSL. The Transformation component was extended to handle EDMM model fragments as well as *inputs* and *outputs* for the determined deployment groups.

The *Orchestration* component is required to determine the deployment order and to invoke the *Orchestration Plugins*. A plugin wraps the interaction with a certain deployment technology, e. g., by utilizing SDKs or CLIs. Essentially, each plugin is a subprocess that executes the DTSM using the given input

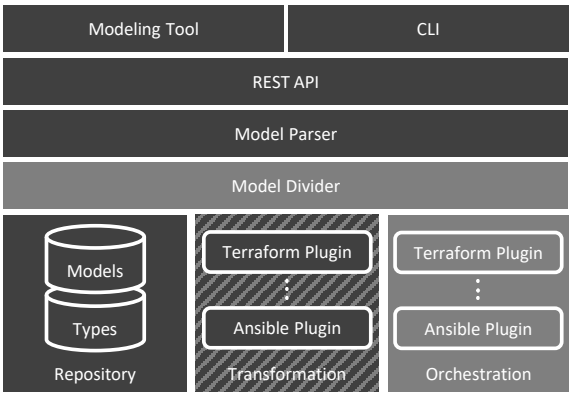

Figure 5: System architecture as an extension of the EDMM Modeling and Transformation System (Wurster et al., 2020a) with new components in light gray and extended components shaded.

data and retrieves runtime information, such as IP addresses, after the successful execution. However, the Orchestration component acts as intermediary and is responsible to populate and store runtime information after a plugin successfully ran. Moreover, it is responsible to determine the input information, i. e., property values and runtime information, required by plugins to successfully execute it. This component is aware of the overall EDMM model, the EDMM model fragments, the deployment groups, and the runtime information propagated back to the system after each deployment task, i. e., Orchestration Plugin invocation.

6 PROTOTYPE & CASE STUDY

In this section, we present a prototypical implementation and demonstrate the usage based on a simplified, but yet effective, case study to validate our approach[2]. The case study scenario is slightly adapted from the one introduced in Fig. 2 because we validate the approach based on more than two deployment technologies. The prototype is based on two major software components: (i) the EDMM Modeling Tool based on Eclipse Winery (Kopp et al., 2013) and (ii) the EDMM Transformation Framework (Wurster et al., 2019a, 2020a). In the course of this paper, we extended the EDMM Framework by the light gray and shaded components as depicted in Fig. 5.

The holistic EDMM model is created using the web-based EDMM Modeling Tool. Users compose the structure by drag-and-drop desired components to the canvas and define respective relations between

[2]A concrete example of the depicted case study including the overall EDMM model and the translated DTSMs is available online at https://bit.ly/37aZDv3.

them by connecting the components. Further, users define configuration properties, such as port numbers, for certain components by directly specifying concrete values or by referencing properties from other components. For example, the "Petclinic" web application needs to connect to the MySQL database at runtime and requires information concerning the endpoint and credentials. Users may define properties referencing properties from other components that have concrete values assigned. For example, a notation like `${<Component>.<Property>}` allows users to reference property values from related components, as depicted by the "DB_User" property of the "Petclinic" component in Fig. 6. Moreover, special runtime properties can be referenced for information that is only available after the deployment of certain components, e. g., IP addresses of VMs.

After exporting the model according to EDMM's YAML specification, the technology-specific regions can directly be define along the deployment model. The `technology_regions` block defines the deployment technology and a list of components that should be deployed with exactly this technology[2].

The transformation into the desired DTSMs is started using the EDMM CLI. The EDMM Transformation Framework parses the given model and the holistic EDMM model is divided depending on the defined technology regions. The result is stored in-memory to execute suitably the corresponding transformation plugin. Each transformation plugin employs the logic to transform modeled components and related artifacts to the files and templates required by a deployment technology. However, we extended the transformation engine and the existing transformation plugins for Ansible, Kubernetes, and Terraform to handle cross-technology property references and to resolve them during transformation. For example, the Kubernetes plugin is able to resolve the database properties and translates them into a "ConfigMap". For runtime information, e. g., IP addresses, plugins that are able to handle "Compute" components, e. g., Terraform, define technology-specific modeling constructs in the DTSM so that such information can be retrieved as *outputs* by the system.

The Orchestration component calculates the deployment order by sorting the deployment groups topologically and invokes the respective Orchestration Plugins. The required *inputs* and *outputs*, i. e., model properties or runtime information, are retrieved, stored, and distributed before and after executing a DTSM. To prove the feasibility, we implemented three plugins, for Terraform, Ansible, and Kubernetes, that are able to execute a deployment fully automatically using our approach and EDMM.

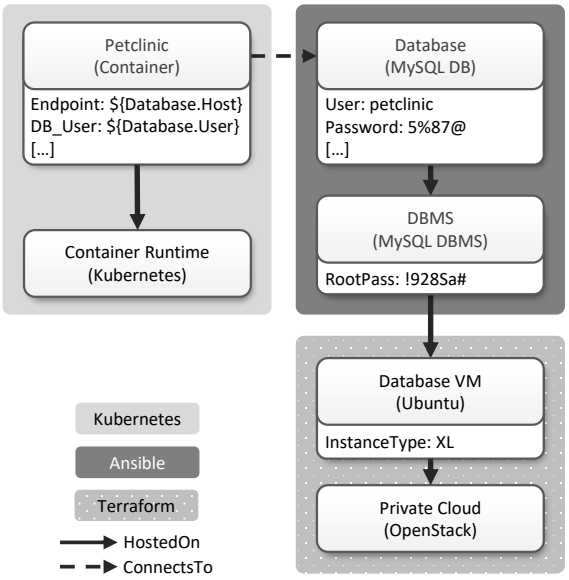

Figure 6: Case study to validate our prototype.

7 DISCUSSION

In practice, different deployment technologies can be integrate in different ways. For example, Terraform already enables that Chef or Puppet agents are automatically installed to pass the necessary information to the management server for registration. However, it relies heavily on the support of a certain deployment technology to support such an integration. Further, this may still result in the need to (i) execute and (ii) coordinate different technologies in a manually determined order. Both is solved in this paper by a technology-independent model-based approach using EDMM and model transformation.

The presented approach is extensible also to further deployment technologies as long as they are compliant with the EDMM metamodel. In general, the plugin-based framework enables the extension with further Transformation and Orchestration Plugins (cf. Fig. 5). At the time of writing, the framework supports the transformation of an EDMM model to 13 different deployment technologies, namely Ansible, Azure Resource Manager, Chef, Docker Compose, Heat Orchestration Template, Kubernetes, Terraform, Puppet, Cloudify, AWS CloudFormation, Salt, Juju, and CFEngine. Further, the framework currently supports Terraform, Ansible, and Kubernetes concerning orchestration. We implemented these three plugins as a first step to validate the overall approach. Even though the presented motivating scenario is indeed simple, it still highlights the challenges to combine and coordinate these deployment technologies.

In contrast to orchestrators that process a single deployment model, references to configuration or runtime properties provided by other components must be explicitly modeled in EDMM because the concrete operations and their implementations supported by the different deployment technologies are not known in advance. Thus, we cannot rely on the deployment processing capabilities for dependencies between components that are deployed by different technologies. By explicitly modeling the required inputs from other components, the deployment of the whole application can be automated without manual user inputs during the deployment.

Further, the presented approach is not only applicable to EDMM and its underlying YAML syntax. TOSCA, for example, is heavily used in research (Bellendorf and Mann, 2019) but barely supported by production-ready deployment technologies. However, previous work (Wurster et al., 2020b) showed and validated that EDMM is a compliant subset of TOSCA, which makes it possible to combine it with our approach to target a wider audience and help to bridge the gap between the academic state-of-the-art and the industrial state-of-the-practice.

The presented system architecture, as most production-ready deployment systems, is based on a central orchestration component that coordinates the deployment of the generated DTSMs. It is not based on a workflow technology and an underlying workflow language such as BPMN or BPEL. However, it could be combined with existing approaches employing the advantages of workflow technologies, either by orchestration workflows or by using a choreography for a decentralized deployment executed by individual independent participants (Wild et al., 2020). This has no influence on the functionality itself, but by using standardized workflow languages we can take advantage of the capabilities of workflow technologies such as scalability, reliability, robustness, and transactional processing.

8 RELATED WORK

The problem of automating the deployment of multi-component applications on cloud platform is well-known (Wettinger et al., 2018), with the OASIS standard TOSCA (OASIS, 2015) being one of the most known approaches in this direction (Bergmayr et al., 2018). TOSCA provides a standardized language for specifying multi-component application in a portable way, and to automate their deployment on cloud infrastructures, provided that the latter support the declarative processing of TOSCA application specifications, e. g., featured by OpenTOSCA (Breitenbücher et al., 2016). Various other approaches follow a similar approach,e. g., CAMEL (Achilleos et al., 2019), MODAClouds (Di Nitto et al., 2017), Panarello et al. (Panarello et al., 2017), SeaClouds (Brogi et al., 2014) and trans-cloud (Carrasco et al., 2018) (just to mention some), by starting from a vendor-agnostic specification of a multi-component application, and enabling its deployment on heterogeneous clouds provided that the latter provide ad-hoc components or middlewares for processing application specifications. Further, Breitenbücher et al. (2013) enable the integration of script-centric and service-centric provisioning and configuration technologies based on Management Planlets, while Wettinger et al. (2013) integrate the usage of configuration management tools with TOSCA. Our approach differs from all those listed above, as we aim at automatically generating the deployment artifacts needed to deploy applications with already existing deployment technologies as they are, i. e., without requiring any additional software component or middleware.

Closer approaches to ours are those by Di Cosmo et al. (2014, 2015), Guillén et al. (2013), and Alipour and Liu (2018). They all share our baseline idea of generating concrete deployment artifacts from a vendor-agnostic specification of an application and of its desired configuration. Di Cosmo et al. (2014, 2015) 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. Guillén et al. (2013) and Alipour and Liu (2018) instead transform an originally vendor-independent application into a platform-specific solution to enact its deployment. The above approaches however differ from ours, as Di Cosmo et al. (2014, 2015) targets only OpenStack-based application deployments, while Guillén et al. (2013) and Alipour and Liu (2018) are intended to process applications whose sources are available to their frameworks.

Other approaches worth mentioning are those by Brabra et al. (2019) and Bogo et al. (2020), which enable deploying TOSCA models with different deployment technologies. Brabra et al. (2019) use model-to-model and text-to-model transformation concepts to transform to different technologies such as Juju, Kubernetes, or Terraform. Bogo et al. (2020) instead propose TOSKOSE, a distributed solution for enacting the deployment of multi-service application on top of Docker Compose and Kubernetes, given the specification of their deployment in TOSCA. Our approach differs from those by Brabra et al. (2019) and Bogo et al. (2020) since we target 13 different deployment technologies (therein included all those

supported by such approaches) and we enable to partition the deployment of the application over different deployment technologies at the same time.

To summarize, to the best of our knowledge, ours is the first approach enabling to split a single application into different parts to be deployed with different deployment technologies, while at the same time not requiring any middleware to be installed. How an application can be split into multiple parts and then deployed independently has been discussed in different works (Sebrechts et al., 2018; Saatkamp et al., 2019; Wild et al., 2020). However, Saatkamp et al. (2019) and Wild et al. (2020) enable an automated decentralized deployment but support only TOSCA. Sebrechts et al. (2018) enables the transformation from TOSCA to Juju models but focuses on the reuse of the deployment knowledge and the partitioning of management tasks instead of supporting different technologies. Such decentralized deployment concepts could be combined with our approach to split and transform to different deployment automation technologies.

9 CONCLUSIONS

In this paper we presented an approach (i) to annotate parts of a holistic deployment model to be deployed with different deployment technologies, (ii) to automatically transform an annotated model into multiple DTSMs each deploying the annotated part of the whole application, and (iii) to automatically coordinate the deployment execution using different deployment technologies. Thus, for each part of an application the best fitting deployment technology can be selected and combined, e. g., depending on the hosting environment or the required capabilities. For the automated deployment, a central orchestration component coordinates the deployment of the DTSMs in the correct order and handles the input and output between the different deployment technologies. Thus, the deployment processing can be completely automated.

The presented prototype supports currently the transformation to 13 different deployment technologies and the deployment execution for three deployment technologies. It will be extended in future by implementing additional transformation and orchestration plugins. Further, we plan to extend the approach also to enable the advantages of using workflow technologies and to achieve a decentralized deployment by employing choreographies. This further increase the flexibility of the deployment of complex distributed applications, especially for application scenarios involving multiple organizational entities, e. g., different departments or companies.

ACKNOWLEDGMENTS

This work is partially funded by the following projects: *RADON* (EU, 825040), IC4F (01MA17008G), and *DECLware* (PRA_2018_66).

REFERENCES

Achilleos, A. P., Kritikos, K., Rossini, A., Kapitsaki, G. M., Domaschka, J., Orzechowski, M., Seybold, D., Griesinger, F., Nikolov, N., Romero, D., and Papadopoulos, G. A. (2019). The cloud application modelling and execution language. *Journal of Cloud Computing*, 8.

Alipour, H. and Liu, Y. (2018). Model Driven Deployment of Auto-Scaling Services on Multiple Clouds. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 93–96. IEEE.

Bellendorf, J. and Mann, Z. Á. (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 Computing Surveys*, 51(1).

Bogo, M., Soldani, J., Neri, D., and Brogi, A. (2020). Component-aware orchestration of cloud-based enterprise applications, from tosca to docker and kubernetes. *Software: Practice and Experience*.

Brabra, H., Mtibaa, A., Gaaloul, W., Benatallah, B., and Gargouri, F. (2019). Model-Driven Orchestration for Cloud Resources. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 422–429.

Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., and Wettinger, J. (2014). Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *International Conference on Cloud Engineering (IC2E 2014)*, pages 87–96. IEEE.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wettinger, J. (2013). Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013)*, pages 130–148. Springer.

Breitenbücher, U., Endres, C., Képes, K., Kopp, O., Leymann, F., Wagner, S., Wettinger, J., and Zimmermann, M. (2016). The OpenTOSCA Ecosystem - Concepts & Tools. *European Space project on Smart Systems, Big Data, Future Internet - Towards Serving the Grand Societal Challenges*, pages 112–130.

Brogi, A., Canciani, A., and Soldani, J. (2018). Fault-aware management protocols for multi-component applications. *Journal of Systems and Software*, 139:189–210.

Brogi, A., Carrasco, J., Cubo, J., D'Andria, F., Ibrahim, A., Pimentel, E., and Soldani, J. (2014). 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)*, pages 758–763. SciTePress.

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

Di Cosmo, R., Eiche, A., Mauro, J., Zacchiroli, S., Zavattaro, G., and Zwolakowski, J. (2015). Automatic Deployment of Services in the Cloud with Aeolus Blender. In *Service-Oriented Computing*, pages 397–411. Springer.

Di Cosmo, R., Lienhardt, M., Treinen, R., Zacchiroli, S., Zwolakowski, J., Eiche, A., and Agahi, A. (2014). Automated synthesis and deployment of cloud applications. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pages 211—222. ACM.

Di Nitto, E., Matthews, P., Petcu, D., and Solberg, A. (2017). *Model-Driven Development and Operation of Multi-Cloud Applications: The MODAClouds Approach*. SpringerBriefs in Applied Sciences and Technology. Springer, Cham.

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.

Guerriero, M., Garriga, M., Tamburri, D. A., and Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 580–589.

Guillén, J., Miranda, J., Murillo, J. M., and Canal, C. (2013). A service-oriented framework for developing cross cloud migratable software. *Journal of Systems and Software*, 86(9):2294–2308.

Herry, H., Anderson, P., and Wickler, G. (2011). Automated Planning for Configuration Changes. In *Proceedings of the 25th International Conference on Large Installation System Administration (LISA 2011)*, pages 57–68. USENIX.

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

Oppenheimer, D., Ganapathi, A., and Patterson, D. A. (2003). 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.

Panarello, A., Breitenbücher, U., Leymann, F., Puliafito, A., and Zimmermann, M. (2017). 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*, pages 194—-201. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST).

Saatkamp, K., Breitenbücher, U., Kopp, O., and Leymann, F. (2019). Method, formalization, and algorithms to split topology models for distributed cloud application deployments. *Computing*, pages 1–21.

Sebrechts, M., Van Seghbroeck, G., Wauters, T., Volckaert, B., and De Turck, F. (2018). Orchestrator conversation: Distributed management of cloud applications. *International Journal of Network Management*, 28(6).

Wettinger, J., Andrikopoulos, V., Leymann, F., and Strauch, S. (2018). Middleware-Oriented Deployment Automation for Cloud Applications. *IEEE Transactions on Cloud Computing*, 6(4):1054–1066.

Wettinger, J., Behrendt, M., Binz, T., Breitenbücher, U., Breiter, G., Leymann, F., Moser, S., Schwertle, I., and Spatzier, T. (2013). Integrating configuration management with model-driven cloud management based on tosca. In *Proceedings of the 3rd International Conference on Cloud Computing and Service Science (CLOSER 2013)*, pages 437–446. SciTePress.

Wild, K., Breitenbücher, U., Képes, K., Leymann, F., and Weder, B. (2020). Decentralized Cross-Organizational Application Deployment Automation: An Approach for Generating Deployment Choreographies Based on Declarative Deployment Models. In *Proceedings of the 32nd Conference on Advanced Information Systems Engineering (CAiSE 2020)*, volume 12127 of *Lecture Notes in Computer Science*, pages 20–35. Springer International Publishing.

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., Brogi, A., Harzenetter, L., Leymann, F., and Soldani, J. (2020a). Technology-Agnostic Declarative Deployment Automation of Cloud Applications. In *Proceedings of the 8th European Conference on Service-Oriented and Cloud Computing (ESOCC 2020)*, pages 97–112. 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*.

Wurster, M., Breitenbücher, U., Harzenetter, L., Leymann, F., Soldani, J., and Yussupov, V. (2020b). TOSCA Light: Bridging the Gap between the TOSCA Specification and Production-ready Deployment Technologies. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 216–226. SciTePress.