The Deployment Model Abstraction Framework

Marcel Weller[1], Uwe Breitenbücher[2], Sandro Speth[1], and Steffen Becker[1]

[1] Institute of Software Engineering, University of Stuttgart, Germany
[2] Institute of Architecture of Application Systems, University of Stuttgart, Germany
`[lastname]@informatik.uni-stuttgart.de`

Abstract. For the deployment of applications, various deployment technologies, such as Kubernetes and Terraform, are available to automate the deployment of applications. However, to use these technologies, developers must acquire specialized knowledge about these deployment technologies to create, maintain, and understand deployment models, for example, configuration files created with Kubernetes. In this work, we present and demonstrate the Deployment Model Abstraction Framework (DeMAF), a tool that enables transforming *technology-specific* deployment models into *technology-agnostic* deployment models that are modeled based on the Essential Deployment Metamodel (EDMM). The resulting technology-agnostic EDMM deployment models express deployments only by using the general modeling concepts that are supported by the 13 most prominent technologies. Therefore, the target audience for this demonstration includes developers and architects, who will be shown that such transformations can be automated and that the resulting EDMM models can be understood without knowledge of the original deployment technology. We evaluate the general practical feasibility of the approach by a case study that demonstrates a scenario based on the T2-Project and the technologies Terraform, Kubernetes, and Helm.

Keywords: Deployment Models · Infrastructure-as-Code · Abstraction · Transformation · Essential Deployment Metamodel.

1 Introduction & Motivation

Most deployment technologies enable executing deployments automatically based on *declarative deployment models*, which describe the application's components to be deployed, their configurations, as well as their dependencies [5]. A deployment model often not only describes the deployment of software components but also needs to cover a variety of other aspects, e.g., the provisioning of infrastructure resources such as virtual machines. As a result, many different deployment technologies with special purposes have been developed, which differ significantly from each other regarding supported (i) *deployment modeling languages* and the (ii) *deployment features* they provide [2, 5]. Moreover, especially for deploying complex applications that consist of various components running in different heterogeneous environments, e.g., a multi-cloud application, often multiple technologies must be combined, which requires deployment models that embed deployment models of other technologies to cover all aspects.

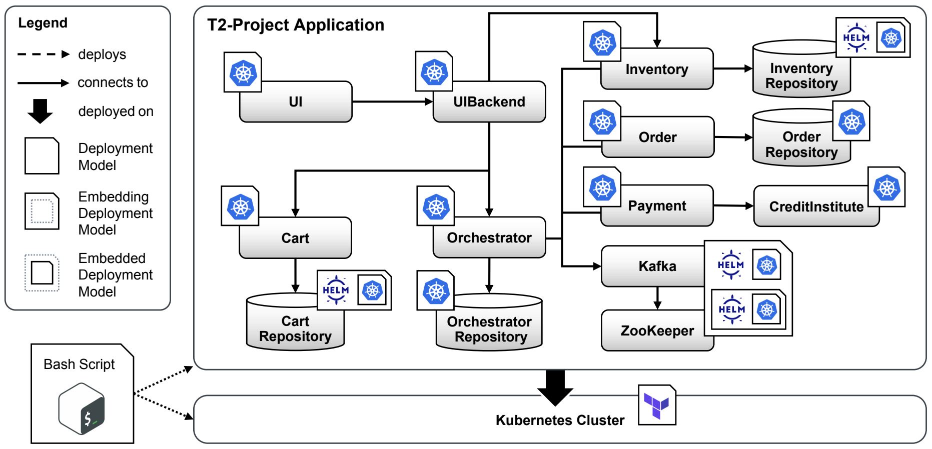

Fig. 1: Overview of the T2-Project and the created deployment model.

The various available deployment technologies contribute to full automation since almost every composition of components can be deployed by one of these technologies or a combination thereof. For example, the *T2-Project*[3] is a reference architecture for applications following the *microservice architectural style* [3]. It comprises several microservices and databases that together implement an e-commerce webshop for tea as shown in fig. 1. It contains, e.g., a *UI Service* and several backend services for providing features such as a shopping cart and online payment. The services communicate through asynchronous messaging realized by the *Kafka Service*. Let us assume we want to deploy this composite application on a Kubernetes cluster. We can model the deployment of all business services directly with Kubernetes configuration files. However, for some services, it makes sense to use other deployment technologies that better fit the requirements. For example, for deploying common components such as database systems, the technology *Helm*[4] is better suited since it provides reusable Kubernetes deployment models called *Helm charts*. Thus, such Helm charts can be easily reused, which eliminates the need to define own Kubernetes configuration files. Helm charts always contain an embedded Kubernetes deployment model of the service that they provide. They can also embed other Helm charts, e.g., the Helm chart for the *Kafka Service* embeds another Helm chart for the *ZooKeeper* service. Since Kubernetes itself only enables to describe software deployments based on containerization, we used the Infrastructure-as-Code technology *Terraform*[5] to model the deployment of the Kubernetes cluster itself. Since multiple technologies are involved, we need to provide an overall deployment model that invokes all deployment technologies with the corresponding models, which we implement as a *Bash script*. All mentioned models are available on GitHub[6].

[3] Implementation of the T2-Project on GitHub: https://github.com/t2-project
[4] Helm: https://helm.sh/ [5] Terraform: https://www.terraform.io/ [6] Deployment model for the T2-Project on GitHub: https://github.com/Well5a/kube

2 Problem Statement

While it makes sense in the scenario described above to select the best fitting deployment technology for each service, it obviously leads to a high number of heterogeneous deployment models and used technologies. Thus, if somebody wants to understand this deployment only based on the provided models, they would have to understand all technical details of all used modeling languages, deployment technologies, and also the overall orchestration. Unfortunately, due to the various available technologies and their technical complexity, the required knowledge for understanding and maintaining such models is immense, especially if they are combined and nested. As a result, an overview of the entire deployment including all services, databases, their configurations, and their dependencies are hard to get from these *technology-specific deployment models*.

3 Deployment Model Abstraction Framework (DeMAF)

In this demonstration, we introduce the *Deployment Model Abstraction Framework (DeMAF)* that addresses the problems described in the previous section by transforming *technology-specific deployment models* into *technology-agnostic deployment models* that are described using the technology-independent Essential Deployment Metamodel (EDMM) [5]. EDMM is the result of a systematic analysis of deployment technologies conducted by Wurster et al. [5] and contains only model entities that can be mapped to 13 of the most prominent declarative deployment technologies such as Terraform, Kubernetes, and AWS CloudFormation. Therefore, EDMM models contain no deployment technology-specific details and only describe the (i) components that get deployed, (ii) their configurations, as well as (iii) their dependencies. Thus, EDMM models provide all information required for understanding the architecture of the deployed system and require no technical expertise about concrete deployment technologies.

An overview of the architecture of DeMAF is shown in fig. 2. It comprises several components that are independent self-contained microservices. For supporting the transformation of technology-specific deployment models in an extensible way, the DeMAF follows a plugin-based approach: Each plugin is responsible for transforming a deployment model created with a specific deployment technology into entities of EDMM. This way, the framework can be extended to support more technologies, new major versions of already supported technologies, or alternative analysis methods that work better or supplement existing plugins.

At startup, the plugins register at the *Analysis Manager*, which manages the transformation process, handles user interaction, and persists information about registered plugins in the *Config Database*. We realized the user interface of the Analysis Manager through a command-line interface that provides a command for inputting a deployment model and starting the transformation process. After that, the Analysis Manager determines the deployment technology of the given deployment model, creates a task, and sends it to the appropriate plugin. The tasks are persisted in the *Task Database* to keep track of their status. A

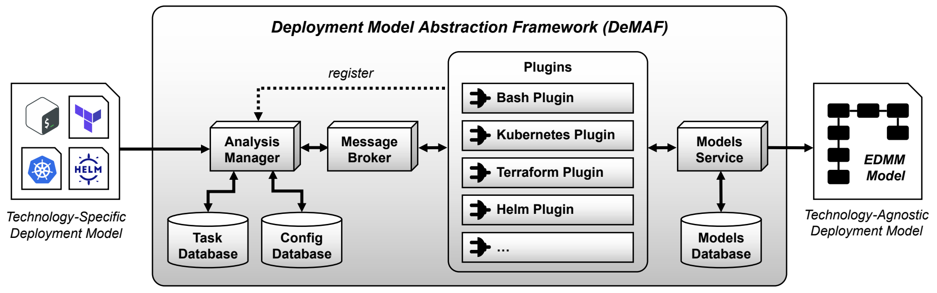

Fig. 2: Concept of the Deployment Model Abstraction Framework.

RabbitMQ *Message Broker* facilitates asynchronous communication between the Analysis Manager and the plugins. To enable the transformation of technology-specific deployment models that utilize several other deployment technologies, the plugins can detect *embedded deployment models* and the corresponding deployment technologies. An embedded deployment model is a deployment model that is contained in another deployment model, an *embedding deployment model*. The embedded deployment model prescriptively describes the deployment of a specific part of the embedding deployment model. A deployment model can embed any number of other deployment models. The embedded deployment model is possibly created with a different technology than the embedding deployment model and can recursively contain further embedded deployment models. The plugins report detected embedded deployment models to the Analysis Manager. The DeMAF analyzes each embedded deployment model separately by distributing tasks for each embedded deployment model across the plugins.

During the analysis, the plugins transform the found information into an EDMM model, which is persisted in the *Models Database* and managed by the *Models Service*, which provides a common interface for all plugins. The transformation process is finished when the whole given technology-specific deployment model has been analyzed, i.e., including all embedded deployment models. The DeMAF then outputs the created EDMM model as a YAML file[7].

We implemented a prototype of the DeMAF[8] that currently supports the four deployment technologies Kubernetes, Terraform, Helm, and Bash. We chose these technologies because they cover a variety of use cases and are commonly integrated in deployment models: Terraform focuses on infrastructure components, while Kubernetes is based on containerization and restricted to the deployment of software components. Helm is a package manager for Kubernetes, which provides reusable Kubernetes deployment models. Bash follows an imperative modeling approach. The plugins are implemented in Java and parse the objects of the technology-specific deployment model into an intermediary representation of Java objects that are then transformed into EDMM entities.

[7] EDMM in YAML Specification: https://github.com/UST-EDMM/spec-yaml
[8] GitHub organization with the DeMAF prototype: https://github.com/UST-DeMAF

4 Evaluation: Case Study

We conducted a case study for validating the DeMAF's ability to transform technology-specific deployment models into EDMM models. We used the deployment model of the T2-Project introduced in section 1 as input, for which we first manually created an EDMM model that contains all information we expect, called the *expected* EDMM model. To ensure that this information is complete, we thoroughly examined the files of the technology-specific deployment model for the T2-Project including all embedded deployment models. We compared this expected EDMM model with the *actual* EDMM model that is automatically generated by the DeMAF and investigated the differences. Thereby, we ignored minor differences in naming, YAML indentation, or the order of elements. The comparison showed that the DeMAF can successfully transform the given technology-specific model into an appropriate EDMM model: The actual EDMM model contains all of the information in comparison to the expected EDMM model. The DeMAF created EDMM components for all services of the T2-Project application shown in fig. 1. It connected these components with appropriate EDMM relations that accurately describe the dependencies between the services. For the Kubernetes cluster, it created several EDMM components that describe the provided resources and other capabilities such as a container runtime. Additionally, the DeMAF created EDMM relations showing how these components host the T2-Project services. All evaluation results including the expected and actual EDMM model are provided on Zenodo[9]. Moreover, we created a video that is available on YouTube which demonstrates this case study[10].

5 Related Work

Previous work already provided concepts for transforming technology-specific deployment models into technology-agnostic deployment models using the *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [1,4]. Wettinger et al. [4] crawl public code repositories for technology-specific deployment models and transform them into more generic representations that can be integrated with each other. They crawl technology-specific deployment models created with Chef and Juju and, for each of them, generate TOSCA node types that contain meta information such as the name. However, they do not transform the technology-specific deployment models but rather attach them to the generated TOSCA node types by wrapping them into TOSCA artifacts. Endres et al. [1] follow a similar approach. They crawl public code repositories for technology-specific deployment models created with Chef and derive the structure of the application that is described into so-called *technology-agnostic topology models*, similarly to our work. These topology models are based on the TOSCA standard and are combined with the originating technology-specific deployment models, which are together stored in a repository. This enables the modeling and deployment of applications composed of several technology-specific deployment models

[9] Zenodo repository with evaluation results: https://doi.org/10.5281/zenodo.6824223
[10] https://youtu.be/nHl-8zxY-mU

without the need to understand the technical details of the used technologies because this information is given by the attached technology-agnostic topology models. However, their tool can currently only transform technology-specific deployment models created with Chef. Moreover, the overall concept does not deal with the transformation of more complex technology-specific deployment models that contain embedded deployment models created with different deployment technologies, which is supported by the DeMAF approach.

6 Conclusion and Future Work

The first prototypical realization of the DeMAF showed promising results on which we can now build on. In future work, we will improve the existing plugins and provide support for further deployment technologies. This may include reworking the transformation logic into a more standardized and verifiable approach, e.g., using the Eclipse Modeling Framework (EMF) to transform the deployment models with transformation rules specified in the MOF QVT (Query/View/Transformation) standard. Additionally, other researchers or developers can use the DeMAF and provide custom plugins. However, developing a system that reliably transforms arbitrary deployment technologies requires substantial effort. Because deployment technologies differ heavily, we would need to implement many plugins which allow the reuse of code to a limited extent. Additionally, deployment technologies evolve by changing existing features or introducing new concepts that make a plugin incompatible. Therefore, it would be beneficial to find more general approaches for analyzing the technology-specific deployment models, like monitoring of network traffic of a deployed application.

References

1. Endres, C., et al.: Anything to Topology - A Method and System Architecture to Topologize Technology-Specific Application Deployment Artifacts. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017). pp. 180–190. SciTePress (Apr 2017)
2. Lu, H., et al.: Pattern-Based Deployment Service for Next Generation Clouds. In: 2013 IEEE Ninth World Congress on Services. pp. 464–471. IEEE (2013)
3. Speth, S., Stieß, S., Becker, S.: A Saga Pattern Microservice Reference Architecture for an Elastic SLO Violation Analysis. In: Companions Proceedings of 19th IEEE International Conference on Software Architecture (ICSA-C 2022). IEEE (Mar 2022)
4. Wettinger, J., Breitenbücher, U., Kopp, O., Leymann, F.: Streamlining DevOps Automation for Cloud Applications using TOSCA as Standardized Metamodel. Future Generation Computer Systems pp. 317–332 (Aug 2015)
5. Wurster, M., et al.: The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies. SICS Software-Intensive Cyber-Physical Systems **35**, 63–75 (Aug 2019)