Using Variability4TOSCA and OpenTOSCA Vintner for Holistically Managing Deployment Variability (Extended Abstract)

Miles Stötzner[1,*], Uwe Breitenbücher[2], Robin D. Pesl[3] and Steffen Becker[1]

[1]Institute of Software Engineering, University of Stuttgart, Stuttgart, Germany
[2]Herman Hollerith Zentrum, Reutlingen University, Reutlingen, Germany
[3]Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany

Abstract

Applications often need to be deployed in different variants due to different customer requirements. However, since modern applications often need to be deployed using multiple deployment technologies in combination, such as Ansible and Terraform, the deployment variability must be considered in a holistic way. To tackle this, we previously developed Variability4TOSCA and the prototype OpenTOSCA Vintner, which is a TOSCA preprocessing and management layer that implements Variability4TOSCA. In this demonstration, we present a detailed case study that shows how to model a deployment using Variability4TOSCA, how to resolve the variability using Vintner, and how the result can be deployed.

Keywords

OpenTOSCA Vintner, Variability4TOSCA, TOSCA, Variability, Unfurl, xOpera

1. Introduction

Modern applications often require a combination of multiple deployment technologies, such as Ansible and Terraform. Moreover, customers typically have different requirements for the deployment of applications, e.g., regarding elasticity or the number of users the deployment has to handle. As a result, applications often need to be deployed in different variants, e.g., one variant is a community edition with limited functionalities and a restricted number of users, whereas another variant is the highly scalable enterprise version [1]. Since modeling each of these variants in separate deployment models would result in a huge number of models to be maintained, we developed *Variable Deployment Models* in previous work [2] and extended the concept in our latest paper [1], which was accepted at the research track of CoopIS 2023. In this demonstration, we show practical details of the case study presented in our CoopIS 2023 paper.

Proceedings of the Demonstration Track at International Conference on Cooperative Information Systems 2023, CoopIS 2023, Groningen, The Netherlands, October 30 - November 3, 2023

*Corresponding author.
miles.stoetzner@iste.uni-stuttgart.de (M. Stötzner); uwe.breitenbuecher@reutlingen-university.de (U. Breitenbücher); robin.pesl@iaas.uni-stuttgart.de (R. D. Pesl); steffen.becker@iste.uni-stuttgart.de (S. Becker)
https://miles.stoetzner.de (M. Stötzner)
0000-0003-1538-5516 (M. Stötzner); 0000-0002-8816-5541 (U. Breitenbücher); 0000-0002-5980-9395 (R. D. Pesl); 0000-0002-4532-1460 (S. Becker)

2. Variability4TOSCA and OpenTOSCA Vintner

TOSCA [3] is a standard for modeling the deployment of applications in a portable manner. Please note that we introduce only a subset of TOSCA. The structure of the application to be deployed is modeled as a *topology template*, which consists of *node templates* and *relationship templates*: Node templates represent components, e.g., a web server or a Java application, while relationship templates represent relations between components, e.g., a *hosting* relation or a *SQL database connection*. Node templates and relationship templates are semantically described by *node types* and *relationship types*, respectively. For example, a node template can have the node type "Java20Application". Moreover, *properties* can be used to specify deployment configurations of node templates, e.g., to specify the HTTP port of a web server to be deployed. Finally, *deployment artifacts* are used to provide the implementation files of a component to node templates, e.g., a binary or a Docker image. These *TOSCA models* can then be executed by TOSCA orchestrators, such as OpenTOSCA, Unfurl, or xOpera.

While TOSCA provides an abstraction layer on top of deployment technologies, applications often need to be deployed in different variants. Of course, for each variant, a separate TOSCA model could be created. However, this quickly results in a huge number of different models that have to be maintained. Therefore, we extended TOSCA to *Variability4TOSCA* [2] in previous work, which was further extended in our CoopIS 2023 paper [1]. Originally, Variability4TOSCA enabled to assign *variability conditions* over *variability inputs* to node and relationship templates that define when the associated template is present in the model [2]. For example, if a Java application needs to be deployed on a server in the community variant but on a scalable PaaS offering for the enterprise variant, both variants can be modeled in a single Variability4TOSCA model by including both potential hosting components and both possible hosting relations but with conditions assigned that specify when which node and relationship template is present. Thus, multiple variants of a deployment can be modeled in a single Variability4TOSCA model. However, we found many practical examples in which also properties and deployment artifacts would benefit from having conditions assigned. For example, if an enterprise variant of the deployment provides significantly more functionalities than the community variant, it makes sense to use different deployment artifacts for them, i.e., different implementations. Therefore, we added the concepts of conditional properties and conditional deployment artifacts [1]. The entire specification of Variability4TOSCA is publicly available at GitHub[1].

OpenTOSCA Vintner [1, 2] is an open-source Variability4TOSCA preprocessor and management layer. Vintner is capable of resolving variability, i.e., evaluating assigned conditions, and enables generating TOSCA models that can be executed by TOSCA-compliant orchestrators, such as Unfurl and xOpera. Vintner has over 350 tests that are executed in an automated pipeline for every pushed commit to attest the implementation. To ensure high code quality, we conduct code linting, code style checks, and code quality analyses. Once a week, a nightly workflow is scheduled that runs full integration tests with Unfurl, xOpera, and Google Cloud Platform (GCP). The source code, an extensive documentation and detailed examples[2] are available on GitHub[3].

[1] https://vintner.opentosca.org/variability4tosca/specification/
[2] https://vintner.opentosca.org
[3] https://github.com/opentosca/opentosca-vintner

3. Demonstration

In this demonstration, we explain the case study of our paper [1] accepted at CoopIS 2023 in more detail and show how it can be modeled technically using Variability4TOSCA. Additionally, we demonstrate how to resolve variability using OpenTOSCA Vintner and how to finally deploy the application. We recorded a video of this demonstration, which is publicly available[4].

3.1. Case Study Description

In previous work [1], we introduced a motivating scenario of a SaaS provider that offers different plans for managed instances of a web shop application, which consists of a shop component and a database. The scenario can be deployed either using the free *community plan* or the paid *enterprise plan*. In the free community plan, the shop component is deployed along with an SQLite database on a virtual machine to minimize costs. Thereby, the shop component is implemented by the community deployment artifact, which implements the core functionality of the shop component and is configured to use the SQLite dialect. In the paid enterprise plan, the shop component is deployed along with a MySQL database on GCP with auto-scaling and high availability enabled. Thereby, the shop component is implemented by the enterprise deployment artifact, which implements the core functionality of the shop but also additional analytical reporting functionalities, and uses the MySQL dialect. With plain TOSCA, two different TOSCA models would be required to describe both variants.

3.2. Using Variability4TOSCA to Model the Case Study

We modeled our case study using Variability4TOSCA. While the complete model is available at GitHub[3], Listing 1 shows an excerpt showing the most important Variability4TOSCA concepts.

For Variability4TOSCA, we introduced *conditional node templates* [2]. In Lines 11 to 13, the MySQL database hosted on GCP is modeled (GCP is not shown for the sake of brevity). The specified condition means that this database is only present in the model if the enterprise plan is selected. The condition is not directly assigned but modeled in the variability options of the model in Line 9 and, thus, can be reused. Thereby, the variability input "plan" is accessed, which is modeled in Lines 4 to 6 and which has the community plan selected by default.

The shop component either connects to the SQLite database or the MySQL database. Therefore, we introduced *conditional relationship templates* [2]. In Lines 17 to 19, the database connection to the SQLite database is modeled with a condition assigned in Line 18 checking if the community plan is selected. Similarly, the database connection to the MySQL database is modeled in Lines 20 to 22 with a condition assigned in Line 21 checking if the enterprise plan is selected. Please note: Technically speaking, the conditions are not assigned to relationship templates but to *requirement assignments* that link node templates and relationship templates, thus, supporting conditional relationship templates that are not explicitly modeled but implicitly.

Since the shop component needs to know the dialect of the database, the "DB_DIALECT" property of the shop component must be set either to "sqlite" or "mysql" depending on which database is present. To model this, we introduced *conditional properties* [1]: In Lines 24 to 26, the

[4] https://youtu.be/6szIGJPuCsU

```
topology_template:
 variability:
  inputs:
   plan:
    type: string
    default: COMMUNITY
  expressions:
   is_community: {equal: [{variability_input: plan}, COMMUNITY]}
   is_enterprise: {equal: [{variability_input: plan}, ENTERPRISE]}
 node_templates:
  gcp_database:
   type: gcp.database
   conditions: {logic_expression: is_enterprise}
  shop:
   type: shop.app
   requirements:
    - database:
       conditions: {logic_expression: is_community}
       node: os_database
    - database:
       conditions: {logic_expression: is_enterprise}
       node: gcp_database
   properties:
    - DB_DIALECT:
       conditions: {node_presence: os_database}
       value: sqlite
    - DB_DIALECT:
       conditions: {node_presence: gcp_database}
       value: mysql
   artifacts:
    - artifact:
       conditions: {logic_expression: is_community}
       file: files/application.community.xz
    - artifact:
       conditions: {logic_expression: is_enterprise}
       file: files/application.enterprise.zip
```

Listing 1: Variability4TOSCA model of the case study (shortened).

"DB_DIALECT" property of the shop gets configured to use the SQLite dialect if the condition in Line 25 evaluates to true. This condition checks the presence of the SQLite database identified by its name "os_database" referring to a database deployed on a virtual machine on OpenStack (OS), which is not shown in Listing 1 for brevity. Similarly, in Line 28, a condition checks for the presence of the MySQL database identified by "gcp_database" to configure the MySQL dialect.

Furthermore, depending on the selected plan, either the community or the enterprise deployment artifact must be deployed. Therefore, as part of Variability4TOSCA, we extended TOSCA by *conditional deployment artifacts* [1]. The community deployment artifact is modeled in Lines 31 to 33 and has a condition assigned in Line 32 checking if the community plan is selected. Similarly, the enterprise deployment artifact is modeled in Lines 34 to 36 with a condition assigned in Line 35 that checks if the enterprise plan is selected.

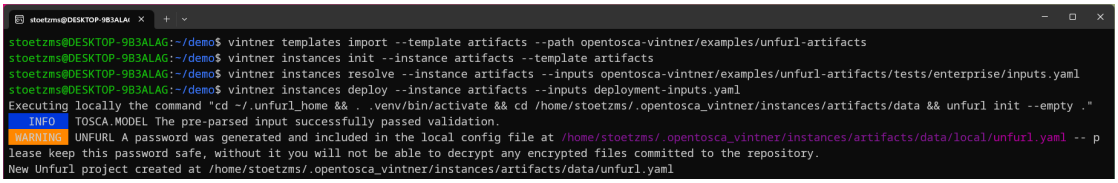

Figure 1: The CLI commands of OpenTOSCA Vintner to deploy the case study.

3.3. Using OpenTOSCA Vintner to Deploy the Case Study

Finally, we deploy the enterprise plan of our case study. We already provided in our CoopIS 2023 paper [1] a step-by-step guide for the deployment[5]. In this demonstration, we extend this guide by a demo video[4] and by the logs of a complete run on our integration pipeline[6].

OpenTOSCA Vintner can be installed on Windows or Linux by using the package managers npm or Yarn. We use the command line interface of Vintner to import the Variability4TOSCA model, as shown in the first command in Figure 1, and initialize an application instance (second command). To resolve the variability, we provide variability inputs stating that the enterprise plan is required (third command). Vintner automatically resolves the variability by evaluating the conditions and derives a TOSCA-compliant model by removing elements whose conditions do not hold. We then deploy this using the fourth command. Vintner passes the resulting model to Unfurl, which uses Ansible and Terraform to deploy our case study as desired on GCP. Therefore, we also need to provide inputs, such as credentials for GCP.

Acknowledgments

This publication was partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) as part of the Software-Defined Car (SofDCar) project (19S21002).

References

[1] M. Stötzner, U. Breitenbücher, R. D. Pesl, S. Becker, Managing the Variability of Deployment Artifacts and Configurations in Deployment Models, in: Cooperative Information Systems (CoopIS 2023), Springer, Accepted.

[2] M. Stötzner, S. Becker, U. Breitenbücher, K. Kálmán, F. Leymann, Modeling Different Deployment Variants of a Composite Application in a Single Declarative Deployment Model, Algorithms 15 (2022) 382.

[3] OASIS, TOSCA Simple Profile in YAML Version 1.3, Organization for the Advancement of Structured Information Standards (OASIS), 2020.

[5] https://vintner.opentosca.org/variability4tosca/guides/artifacts

[6] https://github.com/OpenTOSCA/opentosca-vintner/actions/runs/6100939642/job/16556255878