Executive Summary of the Thesis

Supporting the Development of Infrastructure as Code Using Ansible: a Smart IDE Integrating External Sources

Laurea Magistrale in Computer Science and Engineering - Ingegneria Industriale e Dell'Informazione

Author: Michail Bachras

Advisor: Prof. Elisabetta Di Nitto

Academic year: 2020-2021

1. Introduction

Nowadays, the IT market is dominated by the need to release software quickly and frequently in order to meet the constantly changing needs of customers and users. Consequently, IT organizations were scrambling to find new ways to meet their customers' expectations while remaining competitive and viable. In this dynamic environment, IT enterprises focused on reducing development time and increasing responsiveness by implementing agile and lean methodologies throughout the software development cycle. These methodologies, however, were primarily focused on the development side, ignoring the operations side, resulting in bottlenecks in the development process and delaying software delivery. A new paradigm known as DevOps [1] emerges to bridge this gap and overcome barriers to effective communication between software development and operations teams.

The primary goal of DevOps is to shorten the release cycle and deliver high-quality software on a continuous basis. Therefore, one major requirement is to automate the entire process of configuring infrastructure components in a repeatable manner. Infrastructure as Code (IaC), a fundamental principle of DevOps, is used to implement such an automated process. Furthermore, IaC considers IT infrastructure as software, allowing it to use software principles, methodologies, and tools to speed up software operations. However, several impediments to IaC adoption exist, including a dispersed field of technologies and a lack of support within existing tools. SODALITE [2], a European project, aims to address these issues by creating a development environment that provides the necessary support in deploying and operating complex applications. SODALITE focuses on assisting users in creating TOSCA artifacts that define the overall structure of an application from a deployment standpoint. This approach, however, is insufficient to automate the deployment tasks from start to finish entirely. The reason is the assumption that the implementation scripts used to realize the defined components were written outside of the SODALITE framework. Our contribution aims to fill this gap by complementing the work done in [3]. We enhance the existed development environment with features that assists users in writing Ansible scripts and guides them in relating such scripts to other parts of the deployment specification, such as resource provisioning, thereby facilitating end-to-end model-

ing of a cloud application from the definition of each deployment topology characteristic to the implementation scripts that realize its management and deployment operations.
This paper is structured as follows: Section 2 presents the tools and technologies that have been used and the related state of the art; Section 3 presents the innovative characteristics and the architecture of the system; Section 4 provides the evaluation procedure that we followed and briefly describes the collected results; Finally, Section 5 summarizes the main conclusions of our work.

2. Background

Our work lies in the context of SODALITE and is related to two popular IaC tools, TOSCA [4] and Ansible [5]. Thus, in order to give the reader a clear view of our contribution, we present the main concepts of TOSCA and Ansible, some cutting-edge research projects aimed at facilitating IaC adoption as well as a brief overview of the SODALITE project.

2.1. TOSCA and Ansible

The Topology and Orchestration Specification for Cloud Applications(TOSCA) is an official OASIS standard modeling language whose primary goal is to standardize cloud applications' structure description and automate their deployment and management. TOSCA enables the description of a cloud service's topology along with its operational aspects in a typed topology graph that represents the structure of a cloud application and captures the relationships and dependencies between the application components.
The operational aspects of each node of the application topology are defined within the specification of the related node type and allow for the deployment and management of the respective component. These operations are implemented using artifacts that can be written in a variety of languages, including Python and Chef, but Ansible is used as the implementation language in SODALITE.
Ansible is a free and open-source configuration management tool for provisioning and configuring infrastructure components as well as deploying applications. It employs a Python-based YAML syntax in which the developer specifies the exact step-by-step procedure for bringing the infrastructure to the desired state. Ansible has a significant advantage over other implementation languages in terms of community support, as evidenced by Ansible Galaxy, one of the largest repositories of reusable cloud infrastructure libraries implemented in Ansible.

2.2. Related work

Several research projects have been proposed to support the deployment procedure of an application end-to-end in different programming settings. RADON [6] proposes a DevOps framework that allows developers to effectively manage the entire lifecycle of a complex application that adheres to the serverless and microservices paradigms. ARGON [7] provides a Domain Specific Language that allows users to abstractly represent the needed infrastructure without relying on a specific IaC language. DICE [8] is another model-driven approach that enables users to create language-independent models that are then translated into concrete IaC scripts in the context of Data-Intensive Applications(DIA). Finally, PIACERE [9] is an ongoing project that provides a framework that DevSecOps professionals can use as an end-to-end solution, from modeling the required infrastructure and specifying the functional and non-functional requirements to deploying the application on the defined infrastructure, monitoring it during runtime.

2.3. SODALITE

SODALITE [2] is a research project aimed at simplifying application deployment modeling and execution across multiple, heterogeneous infrastructure resources. It provides a robust toolkit that aids in deploying an application throughout all stages of its modeling procedure. TOSCA and Ansible are key components of this toolset, but not in their pure form. SODALITE provides Domain-Specific Languages that adhere to the principles of TOSCA and Ansible but differ in critical ways to simplify deployment models and increase modularity.
DevOps engineers can define deployment models by using an abstraction of TOSCA that is then translated into TOSCA. Two Domain-Specific Languages (DSLs) have been implemented to accomplish this: Resource Model DSL and Abstract Application Deployment Model DSL. The

provided DSLs decouple the definition of an application model from the resources that it will use. When developing a TOSCA-based model, these concepts are bound together, whereas SODALITE creates two different development environments to support each development procedure separately and more efficiently.
SODALITE also supports the creation of Ansible scripts integrated with the Resource Models. SODALITE provides an Ansible abstraction called Ansible DSL [3], through which developers can define abstract Ansible Models. Ansible DSL adheres to and supports the same conceptual attributes as Ansible, but it groups several attributes that are 'included' in the same semantic category to simplify user interaction and better organize the code. The Ansible Models are transformed into concrete Ansible scripts, which can be associated then with operations of an RM, integrating Ansible with TOSCA.
To support the development of abstract Ansible models, the authors in [3] introduced an innovative user interface called Ansible editor that provides a rich toolset of features, facilitating the development of Ansible scripts.
SODALITE modeling capabilities are delivered to end-users through a user interface known as SODALITE IDE. Users can use textual editors,such as Ansible editor, offered through the SODALITE IDE, which provide a wide range of advanced features that aid in developing the cited models,such as content assistance and validation mechanisms.

3. Ansible support in SODALITE IDE

3.1. Innovation

Having the initial version of the Ansible editor as a starting point, we have expanded the Ansible editor's capabilities and focused on providing valuable features that streamline the development workflow. For example, autocompletion, error messages, and code suggestions are extremely useful for the user and significantly speed up the development process because he/she has all of the necessary information in a single location without having to search through lengthy documentation. This is especially important in the case of Ansible, where Ansible collections, modules, and roles are dispersed across multiple repositories, requiring the user to strain for the desired information. As a result, our primary focus has been to connect the SODALITE IDE with information sources that bring the Ansible content closer to the user and provide him/her with valuable suggestions and constant feedback.
Having this in mind, we created a database containing Ansible content from Ansible Galaxy that can be utilized directly from the Ansible editor, providing a wide range of features. More specifically, the Ansible editor assists the end-user in selecting Ansible collections and modules to import into the model without having to search Ansible Galaxy for them. In addition, the Ansible editor provides content assistance for each Ansible module's parameters, emphasizing the required ones. It also informs the user about inserting values for each parameter by displaying the value type that each parameter expects and presenting the acceptable values and the official description that helps the user understand its purpose. Moreover, the Ansible editor's content assistance offers suggestions for standalone roles uploaded on Ansible Galaxy and Ansible roles included in Ansible collections.
A significant extension we have introduced based on the content of the database is the set of multiple validation mechanisms that check Ansible models for validity issues and provide clear and meaningful recommendations to the end-user on how to fix them. For example, the Ansible editor notifies the users if an Ansible collection name has the incorrect format and instructs them how to correct it. Such errors cause issues during the deployment and management of the cloud application and necessitate a significant amount of time to identify the mistakes in the source code, avoiding the repetitive execution of the Ansible script before fixing all the defined errors. As a result, accurate error messages, accompanied by quick fixes whenever possible, can save the end-users time, increase their productivity, and boost users' satisfaction. The Ansible editor provides the following validation mechanisms:

- Check for missing required parameters of a module
- Inform the end-user about collection and module names that have the wrong format
- Check for acceptable values for each parameter

- Inform the user about collections and roles that are not supported by SODALITE
- Perform type checking for the inserted values of each parameter
- Identify Ansible code smells and informs user for potential solutions

Another key feature of our work is the set of mechanisms that help the user define an Ansible model and import the generated scripts into the corresponding Resource Model. Following the definition of a Resource Model, the Ansible scripts that implement its TOSCA operations must be imported into the RM. As a result, we developed a standardized procedure that guides the user through the definition of the appropriate Ansible model, followed by the generation and integration of the corresponding Ansible script into the RM. In this manner, the user is not lost among the various DSLs that SODALITE IDE provides but instead remains in a specific chain of activities. One critical point to highlight here is that our work does not alter the transparency property of the Ansible script's origin. This property enables the user to select an Ansible script written with an external editor for a TOSCA operation rather than firstly defining an abstract Ansible model (i.e., .ans file) and then generating the concrete Ansible script. Thus, the only requirement for importing an Ansible script is to specify the script's local path in the RM without restricting development to our Ansible editor.

However, if a user decides to develop the related Ansible scripts of a Resource Model using SODALITE IDE rather than an external editor, we have enhanced the Resource Model editor to provide users with clear guidelines on which Ansible files should be created. The user can generate the abstract Ansible model and the corresponding Ansible script for each specified operation within the Resource model for further development. As a result, the user has a clear idea of which Ansible models should be created and for which operations, which is critical when dealing with a complex Resource Model with many interfaces and operations.

Finally, The Ansible editor has been extended to support the communication and information exchange between the provided user interface and Knowledge Base [10], an ontology database that captures TOSCA models' structural and semantics relationships, where developers can upload their work. The user now has access to the contents of the Knowledge Base and can retrieve information for stored Resource Models. This information includes node types, TOSCA interfaces, TOSCA operations, and inputs. In this manner, the user can develop an Ansible model in the context of a resource type stored in the Knowledge Base. Furthermore, the inputs that have been defined in the stored Resource Model can be used as variables in the developing Ansible Model.

Compared to state-of-the-art, the SODALITE Ansible editor is the only one we are aware of that supports an integrated usage of Ansible as the language to define TOSCA operations. Moreover, it is the only one that guides the user in using the modules made available on Ansible Galaxy. These features enable the end-users to reduce their development effort, thus allowing them to use their time and resources in other activities and saving costs.

3.2. Architecture

Figure 1 highlights the elements of the Ansible editor and its interaction with the neighbor components of the SODALITE ecosystem. The Ansible editor is integrated into the SODALITE IDE as an Eclipse Plugin built with Xtext and uses an Ansible abstraction known as Ansible DSL, which allows the users to create abstract Ansible playbooks that are then translated into concrete Ansible scripts. This Ansible Eclipse Plugin is based on the Resource Model Eclipse Plugin, which includes a DSL and a user interface for defining TOSCA reusable entities. Ansible editor provides various services by interacting with three different software components: Semantic Reasoner, Knowledge Base, and the MongoDB database, either implicitly or explicitly.

The semantic Knowledge Base (KB) is a semantic repository containing structured data from SODALITE various domain aspects. SODALITE users can store and retrieve domain models from the KB, which are internally represented as RDF-based knowledge graphs. The Semantic Reasoner, an intermediary between the Knowledge Base and the SODALITE IDE, provides access to this knowledge. In addition, the Ansible editor communicates with the Semantic Reasoner through a REST API called *Semantic Reasoner API*, which allows the editor to retrieve information about Resource Models stored in the Knowledge Base. This information can be the various Resource Models stored in KB, the TOSCA interfaces and operations defined within a specific Resource model, and the data that each interface and operation inputs to the appropriate implementation script to perform the necessary tasks. The Ansible editor presents such information to the end-user to allow for the coherent and consistent development of Ansible models within the context of the chosen resource type and TOSCA operation. Moreover, the Ansible editor communicates with the Defect Predictor via the *Defect_ Predictor API* to detect and present to the user potential code smells induced within the Ansible script.

The MongoDB database delivers Ansible content to Ansible editor end-users, including Ansible collections, modules, and roles gathered from various repositories in the Ansible Galaxy. To facilitate the development of an Ansible model, the Ansible editor communicates with the MongoDB database endpoint and serves the collected Ansible content to the end-user. Figure 2 depicts a detailed schema of the MongoDB database. There are two MongoDB collections in the database: *Ansible_ Galaxy_ Collections* and *Standalone_ Ansible_ Galaxy_ Roles*. *Ansible_ Galaxy_ Collections* contains information about Ansible collections uploaded to Ansible Galaxy, including modules and roles. The database includes the taking parameters and their details for each module, such as type, description, available value choices, and default

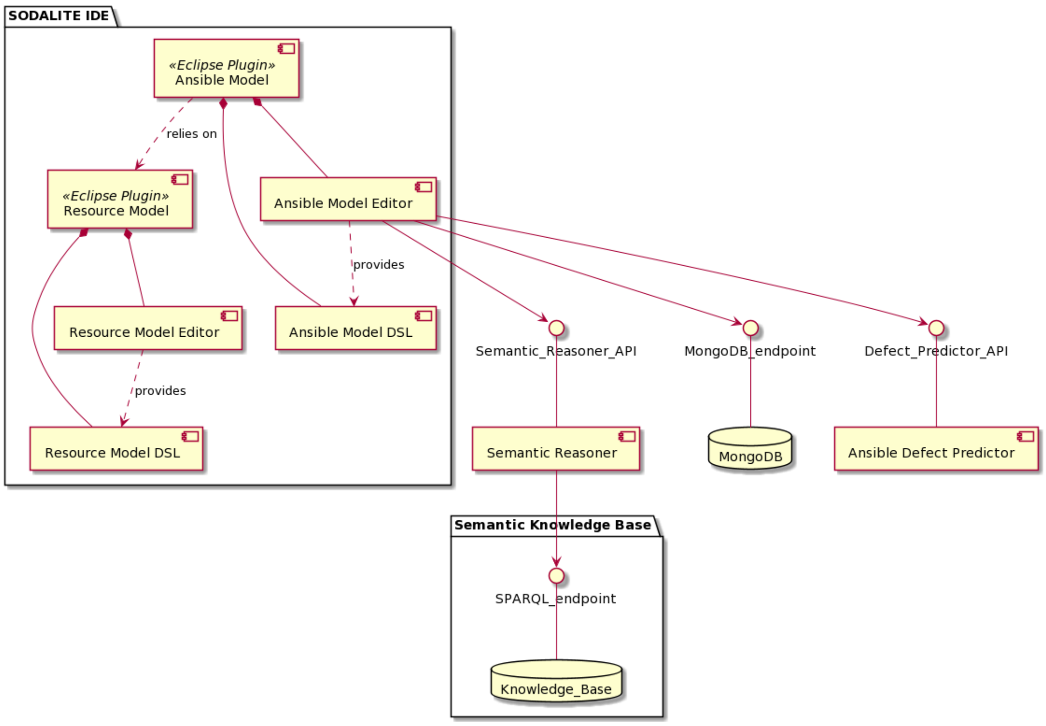
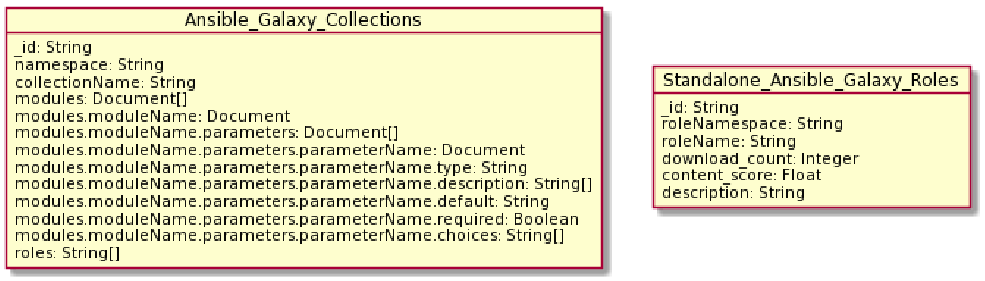


Figure 1: Architecture



Ansible_Galaxy_Collections
_id: String
namespace: String
collectionName: String
modules: Document[]
modules.moduleName: Document
modules.moduleName.parameters: Document[]
modules.moduleName.parameters.parameterName: Document
modules.moduleName.parameters.parameterName.type: String
modules.moduleName.parameters.parameterName.description: String[]
modules.moduleName.parameters.parameterName.default: String
modules.moduleName.parameters.parameterName.required: Boolean
modules.moduleName.parameters.parameterName.choices: String[]
roles: String[]

Standalone_Ansible_Galaxy_Roles
_id: String
roleNamespace: String
roleName: String
download_count: Integer
content_score: Float
description: String

Figure 2: Database schema

value. *Standalone_Ansible_Galaxy_Roles* is a MongoDB collection that contains information about the standalone roles that have been uploaded to Ansible Galaxy. For each role, the database includes details such as the number of downloads, the user satisfaction score, the role description, and how a user can refer to each role (role name, role namespace). The collected Ansible content was chosen based on specific criteria such as the quality of the documentation provided by the developers, the popularity of each project, and the satisfaction among the users as it is reported on Ansible Galaxy.

4. Evaluation

We evaluated our contribution by conducting controlled experiments in which users were asked to create four Ansible playbooks with varying characteristics using our Ansible editor and an external text editor that supports Ansible, named Atom. Our goal was to gather feedback and empirically measure user satisfaction in four different factors [11] that evaluate different aspects of each editor and the provided language. The responses revealed that, when compared to Atom, our Ansible editor required 20% less time to develop the requested Ansible playbooks. Furthermore, when compared to Atom, the testers gave very positive satisfaction scores in the features that improve the editor's usability and assist users in avoiding code errors, and they found the integration of our Ansible editor with TOSCA to be extremely valuable, as it enables the direct exchange information with the related TOSCA models. As a weak point, the testers stated that installing the Ansible editor and the corresponding MongoDB was quite time-consuming when compared to the Atom editor, which only requires the installation of a software package.

5. Conclusions

In this paper, we presented the contributions we made to augment the Ansible support provided from the SODALITE framework. We overviewed the knowledge sources that we exploited to offer a palette of advanced features that facilitate users in developing quality Ansible playbooks via the Ansible editor. Moreover, we conducted an empirical evaluation to examine how our extensions affected developers of Ansible and concluded that our work has a positive influence on the development efforts compared to an external text editor with Ansible support.

References

[1] "What is devops?," 2021.

[2] E. Di Nitto, J. Gorroñogoitia, I. Kumara, *et al.*, "An approach to support automated deployment of applications on heterogeneous cloud-hpc infrastructures," in *SYNASC*, pp. 133–140, 2020.

[3] E. Imperiali, "Providing high-quality support for ansible development," Master's thesis, Politecnico di Milano, 2020.

[4] OASIS, "Tosca simple profile in yaml version 1.3," 2020.

[5] R. H. Inc., "Ansible documentation," 2020.

[6] G. Casale, M. Artač, W.-J. Van Den Heuvel, *et al.*, "Radon: rational decomposition and orchestration for serverless computing," *SICS Software-Intensive Cyber-Physical Systems*, vol. 35, no. 1, pp. 77–87, 2020.

[7] J. Sandobalin, E. Insfran, and S. Abrahao, "An infrastructure modeling approach for multi-cloud provisioning," 2018.

[8] M. Artac, T. Borovšak, *et al.*, "Infrastructure-as-code for data-intensive architectures: A model-driven development approach," in *2018 IEEE ICSA*, pp. 156–15609, IEEE, 2018.

[9] J. Alonso, C. Joubert, *et al.*, "Piacere: Programming trustworthy infrastructure as code in a secure framework," CEUR-WS, 2021.

[10] G. Meditskos, Z. Vasileiou, *et al.*, "A pattern-based semantic lifting of cloud and hpc applications using owl 2 metamodelling," 10 2020.

[11] F. Hermans, M. Pinzger, and A. Deursen, "Domain-specific languages in practice: A user study on the success factors," MODELS '09, (Berlin, Heidelberg), p. 423–437, Springer-Verlag, 2009.