A Model-Driven Solution for Automatic Software Deployment in the Cloud

Franklin Magalhães Ribeiro Junior[1], Tarcísio da Rocha[2], Joanna C. S. Santos[3]**, and Edward David Moreno[2]

[1] Universidade do Tocantins (Unitins), Palmas, TO, Brazil, franklin.mr3@gmail.com
[2] Departamento de Computação, Universidade Federal de Sergipe, São Cristóvão, SE, Brazil
[3] Department of Software Engineering, Rochester Institute of Technology, Rochester, NY, USA

Abstract. Through virtualization, cloud computing offers resources that reduce the costs in the institutions that use hardware and software resources. In this paper, we present a model-based approach to automatically deploy software in the cloud. To evaluate our approach, we conducted an experiment in an IT company in which their software developers used our solution instead of manually deploying software in the cloud. After that, they answered a survey, so we could investigate the following metrics: maintainability, learnability and reduction of developer's workload to deploy software services. The results showed that our solution presented a positive impact of at least of 25% percent for all the metrics. Moreover, since our approach relies upon UML models, it requires less effort to deploy the services as well as it can be used by any professionals that have basic skills about UML.

Keywords: cloud computing; automatic software deployment; model-driven deployment; UML

1 Introduction

The resources offered by cloud computing decrease the costs associated with high data processing demands [13]. Cloud resources have been applied in many fields of knowledge (such as as astronomy, genetics and chemistry) that use exhaustive search algorithms to decode certain structure types [4]. Through virtualization, which abstracts the cloud physical layer in order to offer hardware resources according to demands, cloud computing also has an approach based on Software as a Service (SaaS) [18]. In this approach, clients can request a software service in the cloud provider, which was deployed by the provider itself, or deploy their own service in the cloud [16].

Since cloud environments have their own software architecture, the deployment of a software system in the cloud requires the reconstruction of many

** The author is sponsored by CAPES Brazil to pursue a MS at RIT

existing requirements [4]. Moreover, in environments in which the software deployment in the cloud is costly, introducing the automatic deployment of services and applications would reduce the developer's workload and, as such, it brings advantages.

Thus, in this work we developed and analyzed a model-driven solution to automatically deploy software in the cloud. This solution allows not only software developers, but anyone who has basic skills in UML (Unified Modeling Language) [15], to deploy software in the cloud with a high-level abstraction.

This paper is organized as follows: in Section 2 we explain the requirements needed to be addressed when deploying software in the cloud; in Section 3 we discuss the related work; in Section 4 we elaborate on our proposed solution; in Section 5 we explain the solution's architecture; in Section 6 we show the results we obtained by applying the solution into an IT company; lastly, in Section 7, we make our final considerations about this work.

2 Deploying Software in the Cloud

For a software to properly execute in a cloud architecture, it is required to deploy not only the software itself but also its dependencies (i. e., other services that the software depends on when it is running) [8, 18]. To fulfill this requirement, each developer, who needs to deploy an application in the cloud, has to acquire an access key to the cloud provider, select a machine instance in the provider that supports the application as well as configure and install the virtual machine. After that, the developer needs to deploy the application and its dependencies (in a proper order) in the selected virtual machine [2, 4].

A more efficient way to deploy software in the cloud environment is through automating that process, since the configuration details could be abstracted at a higher level thereby decreasing the human efforts to perform it. Another advantage of the automatic deployment of software in the cloud is the elimination of manual tasks which are prone to error such as: software stack configuration, authentication between virtual machines, specification of dependencies between service components and between services, definition of temporal and spatial dependencies and, lastly, analysis of the resources in the cloud environment in order to identify whether the node fulfills the minimum requirements of the service [11].

3 Related Work

After an extensive investigation of the current state-of-art of approaches for automatic software deployment in the cloud, we selected some of them to describe and analyze. For example, the solutions presented in Vega [6], Wrangler [10] and Disnix [3] used manual codification through scripts, which required more time to deploy the software. Besides that, according to [19] and Fazziki et. al. [9] the disadvantage of such deployment approach is that it increases the costs associated with the codification and human efforts.

We also noted that D&C Based [4], SDO [12] and User-Level Deployment [20] automated the deployment of services in the cloud with the use of programming languages. Comparing to the approaches presented in Vega [6], Wrangler [10] and Disnix [3], those approaches are more advantageous in terms of the time effort in the deployment process.

In addition, Virtual Models [11] and Disnix [3] presented semi-automated mechanisms for deployment. In other words, these approaches still require that certain steps are performed manually during the deployment process.

The solution proposed by Ardagna et. al. [1] presented an approach partially models oriented and semi-automated for software deployment. It requires some level of comprehension from the final user about the details of the cloud structure and a heavy information load in the models for deployment demanded to the user.

The solution presented in this work is based on models for deploying software in the cloud automatically. Our solution only requires the developer to have knowledge about the access key and service provider name, since the specific details are abstracted. The goal is to deploy the software at a higher level of abstraction in order to reduce the human efforts and the time spent in performing the deployment tasks, since the model-based approach is a better way to increase the developer's productivity [9].

4 Proposed Solution

This section explains how the deployment models were defined and adapted to our solution as well as the solution's architecture.

4.1 Solution Workflow

This solution proposes that the developers have the services they need to deploy as well as to create the software deployment UML models. These models define all the information required for the deployment (virtual machines, services, applications, dependencies, operating systems of the virtual machines, services repositories and virtual machines, databases, services provider and access key) as parameters without the need of coding of the configuration of any virtual machine.

After the creation and definition of the models with their respective input parameters, the developer provides them as an input of the system. This triggers an automatic deployment, which encompasses four steps: two for interpreting the UML models, one for creating the software stack and another for automatically generating code. Regarding the steps related to UML models interpretation, the first one is about interpreting a specific model (which contains the elements related to the cloud provider as well as the repository) and the second one is the interpretation of the general model (that has the elements for specifying virtual machines, operating system, services, databases, dependencies and applications).

The software stack is defined as the stack of dependencies between services, which were discovered in the interpreting phase of the general model.

Hence, firstly, the system interprets the general model by collecting data within a file which has a format similar to XML and contains the information specified by the developer in the model. During this collecting process, the system instantiates a list of objects of virtual machines, dependencies, operating systems and services. Subsequently, during the software stack creation step, the system binds the services to the corresponding virtual machines. After that, it individually associates the dependencies with the services in each virtual machine. When these two steps are completed and all data about virtual machines and services were collected, then the step of interpreting of specific models is started. In this step, an XML-like file (.uml format) is read in order to collect data and, finally, automatically generate the code to deploy the software in the cloud.

4.2 Deployment Models Definition

The solution has two UML deployment diagrams as inputs: the *general model* (which is independent of a cloud service provider) and the *specific model* (which has the particular aspects related to the cloud infrastructure). Figures 1a and 1b shows examples of the general model and specific model, respectively. In the example model shown in Fig. 1a there are all the necessary features (services, dependencies between services, operating system, virtual machine, application, databases and optional fields for each service, such as service version, the service directory, etc.) for the software deployment independent of the cloud provider. The features exposed in these general models are used by the specific model (Fig. 1b) to collect the necessary information for the effective implementation of the software in the cloud, such as: access key, virtual machine service repository and the VM instance in the cloud.

Therefore, for an effective modeling of the specific model, developers need to create the general model first, since the specific model needs the information of the virtual machine. Such relationship between the specific and the general model can be noticed in the example shown in Fig. 1b in which the Virtual Machine node corresponds to the same node represented in the example shown in Fig. 1a.

When designing a practical solution to be applied on a large scale environment (with multiple VMs), considering that the developer needs the deployment of multiple applications, the use of models can decrease the work effort to deploy software by reducing coding tasks and increasing the deployment productivity. Hence, our solution allows the creation of multiple VM nodes in the solution model (providing scalability).

5 Solution Architecture

The implementation of the proposed architecture is divided into three views: system, local and remote. This solution is composed of five modules: *Controller*,

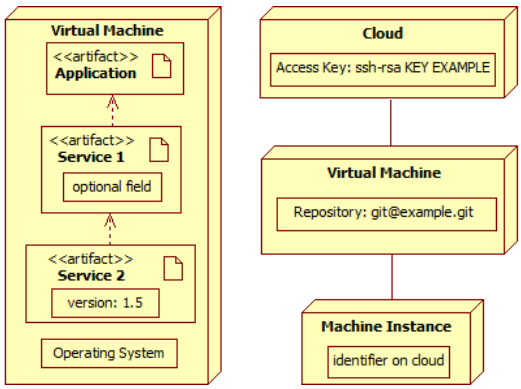


Fig. 1. (a) General model. (b) Specific model

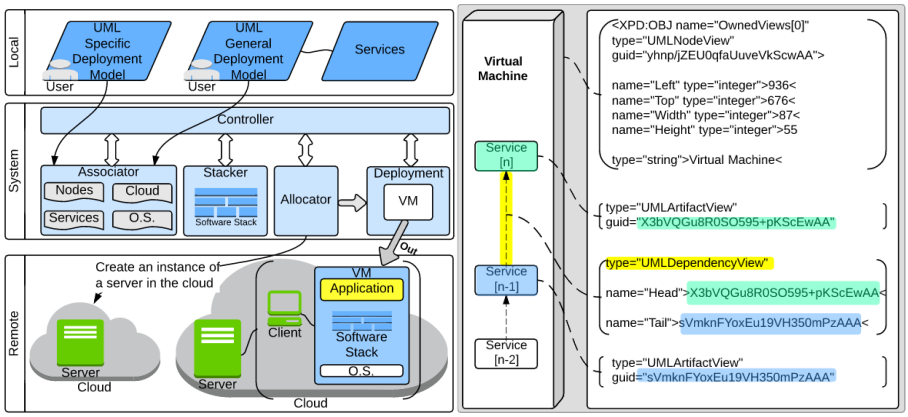
Associator, *Stacker*, *Allocator* and *Deployment*. The Fig. 2a shows the three views of the proposed system architecture for model-based software deployment in the cloud (i) System: showing the five modules of the system, (ii) Location: corresponding to the user's view (which uses the UML deployment diagrams as inputs) and (iii) Remote: it includes the creation of a Chef [5] server instance in the cloud and the final stage of deployment, corresponding to the software installation in the cloud.

5.1 Controller Module

The controller module manages the four other modules, acting as an intermediary in the communication between them. It is responsible for (i) sending a list of services obtained from the Associator to the Stacker and a list of dependency relationships between services (software stack) from the Stacker to the Deployment Module, (ii) enabling communication of the Allocator with the cloud, (iii) notifying the Allocator upon the server instance creation in the cloud, (iv) informing to the Deployment Module which operating system must be installed on the VM, (v) reporting to the Deployment Module which software stack should be allocated to the respective VM and (vi) managing the order of information transmission between the modules and the tasks performed.

5.2 Associator Module

The Associator Module is responsible for interpreting the data within UML models. Its execution routine is divided into two stages: (i) the General Model interpretation and (ii) interpretation of the Specific Model, in which the input files (metadata from UML models) are read line by line at each stage. For each line read in the file, the algorithm within this module searches for the



(a) (b)

Fig. 2. (a) Software deployment solution architecture. (b) Example of identifying the attributes of a model

substring "OwnedViews", which can refer to either a UML component with the "artifact" type (service), or a node (VM), or an association or dependency. When the searched substring is found, the method search for another substring in the same line of the file which can be: (i) "UMLNodeView" (represents a node), (ii) "UMLArtifactView" (representing an artifact), (iii) "UMLAssociationView"(indicating an association and used in the interpretation of the specific model) and (iv) "UMLDependencyView" (representing a dependency). Fig. 2b shows an example of this metadata. Once the attributes related to the UML components are collected, the algorithm instantiates objects with their attributes (GUID, types, and coordinates) and creates lists of these objects for each type of UML component (such as list of node objects, virtual machines, cloud providers, instance machine, services, operating systems and attributes of services).

5.3 Stacker Module

To define the software stack and specify the services belonging to their respective virtual machines, we developed the Stacker module. This Stacker is divided into three steps. In the first step, we establish the dependencies to the services through the two lists of objects obtained from the Associator module. The algorithm of this module then scans the entire list of objects for each dependency artifact (service). For example, when the "guid" identifier (see Fig. 2b) of the service object with an x index from the list is the same as the *head* attribute identifier of an object (dependence) in the dependency list, an array of attributes of the service x will receive the attribute *tail* of the object dependency. With that, the x service will get the service it depends upon to run. The reverse steps of this process are performed for obtaining the identifier, e.g., a parent service

which depends on a child service, which stores the GUID (id) of the parent service in a variable named *previous* (used later to set up the software stack).

In the second step we establish the relationship between the services and virtual machines. To do so, we implemented a method that scans a list of virtual machines (nodes) for each service (device). For each service, it is verified if the service is part of the VM node envelope (if the X and Y coordinates of the service are contained in the envelope of the node).

After the second step, each virtual machine will have a list of services with their dependencies, but unordered. Thus, in the third step we create each software stack for each virtual machine. For that, we developed an algorithm based on a topological sorting algorithm that sorts a graph [7]. Our algorithm creates a graph whose nodes of this data structure are the services objects, in which there is a variable named *previous* for each service that contains the value of the identifier (ID) of the parent service. With these variables that indicate the respective *parent* and *child* services, it is possible to assemble the software stack for each VM (when the graph is mounted, the software stack is ready) by doing a topological sort.

5.4 Allocator Module

The Allocator module performs the allocation of cloud provider(s) into virtual machines and the establishment of machine instances (machine image) in the cloud with the virtual machines that should be deployed. This allocation will make it possible to perform deployment later.

5.5 Deployment Module

In this module, two output files are automatically generated for each virtual machine modeled by the developer in the General and Specific Model. The first file corresponds to a deployment script containing the executable code for deployment. The second one has the application rules and services. The code generated for the deployment was written by the Deployment Module in Ruby language and follow the standards of the Chef platform [5].

We implemented the generation of the deployment executable file in five parts: (i) the creation of the deployment file, (ii) the file header generation, containing the path of the services' settings ("cookbooks"), (iii) the creation of names and definitions of services and the application, (iv) a call to the rules of services and application (which are defined in another file that will be generated by the module) and (v) the executable code deployment, which is comprised of the cloud specification, image VM, operating system, software stack and application.

The generation of the application rules was also implemented in five parts: (i) the creation of the related to the rules of the services, (ii) the file header, consisting of the application name and an indicator that it represents the rules, (iii) the file body with the application name, the repository of the application and services, access key to the cloud provider and the application database 'url',

(iv) optionally, the services' rules containing, for example, the version, port, host, user and password of each service and (v) the classification of each service and the applications that will run on the virtual machine, in the order they are in the software stack.

6 Experiment

We conducted an experiment to evaluate our solution with respect to usability, learnability and the efforts to deploy software in the cloud, according to the point view of developers of software services from industry. To do so, we asked developers from an IT company to deploy an e-commerce software. Thus, these developers created models (General and Specific ones) containing all the necessary requirements for the application deployment.

6.1 Metrics

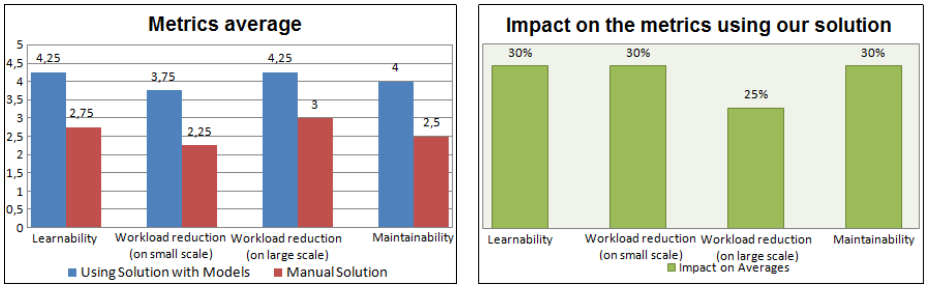
In this experiment, we defined objective assessment metrics, since evaluating software in this scope is subjective. Among the software evaluation metrics found in Nielsen [14] and Santos [17], we selected for this study: (i) Learnability (ease of learning), (ii) Workload and (iii) Maintainability.

6.2 Preparation and Experiment Execution

We asked the participants to deploy a software service in the Amazon Web Services (AWS) cloud provider using the software solution developed in this research. The participants were trained on how to use the model-based solution on performing the assigned task. Moreover, they were told that they would use StarUML as an auxiliary tool to design the General and Specific models. After completing the deployment assigned task, participants filled a survey. This survey aimed to collect data related to the metrics previously mentioned with answers in a Likert scale ranging from 1 (totally agree) to 5 (totally disagree).

6.3 Results and Analysis

After performing the experiment, we collected and analyzed the data. For each metric, we obtained the values shown in Fig. 3a. As shown in this figure, we observed that all the average metric values of learnability, workload reduction on large and small deployment scales and maintainability capacity were higher for those that used the solution. It means that there was a positive impact with the use of the solution proposed by this research. Fig. 3b shows a graph that contains the percentage difference of the weights assigned by the average of each participant, regarding the use and non-use of the model-based solution for software deployment in the cloud. We observed that there was a positive impact in the participants of 30% with respect to each metric of maintainability, learnability and reduced workload in small scale of deployment. Regarding the workload reduction for large-scale implementation, the perceived positive effect was 25%.



(a) (b)

Fig. 3. (a) Metrics average with/without using our solution. (b) Impact on metrics related to the usage of the solution

7 Conclusion

This research presented a model-based approach for automating software deployment in the cloud by using UML models. Our main goal was to reduce the workload for developers and researchers who need to use cloud resources. To evaluate our solution, we conducted an experiment involving a real usage in an IT company. As a result of the experiment, we noticed that there was a reduction in the average workload of up to 25% for developers, apart from the positive impact of 30% increase in learnability metrics and maintainability of the software deployment solution. As a future work, we aim to develop a custom graphical user interface (GUI) for the design of specific UML models to the cloud in order to make modeling even easier for the user and to overcome the results obtained in this research regarding learnability. We also expect to investigate a way to allow the solution to automatically deploy in the same virtual machine in multiple cloud providers using the same model, including hybrid clouds.

References

1. Ardagna, D., Di Nitto, E., Casale, G., Petcu, D., Mohagheghi, P., Mosser, S., Matthews, P., Gericke, A., Ballagny, C., D'Andria, F., et al.: Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In: Proceedings of the 4th International Workshop on Modeling in Software Engineering. pp. 50–56. IEEE Press (2012)
2. Armstrong, D., Djemame, K., Nair, S., Tordsson, J., Ziegler, W.: Towards a contextualization solution for cloud platform services. In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. pp. 328–331. IEEE (2011)
3. Van der Burg, S., De Jonge, M., Dolstra, E., Visser, E.: Software deployment in a dynamic cloud: From device to service orientation in a hospital environment. In: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing. pp. 61–66. IEEE Computer Society (2009)

4. Cała, J., Watson, P.: Automatic software deployment in the azure cloud. In: Distributed Applications and Interoperable Systems. pp. 155–168. Springer (2010)
5. Chef Software, I.: Chef - code can. https://www.chef.io/, (Visited on 01/06/2016)
6. Chieu, T., Karve, A., Mohindra, A., Segal, A.: Simplifying solution deployment on a cloud through composite appliances. In: Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. pp. 1–5. IEEE (2010)
7. Cormen, T.H.: Introduction to algorithms. MIT press (2009)
8. Dudin, E., Smetanin, Y.G.: A review of cloud computing. Scientific and Technical Information Processing 38(4), 280–284 (2011)
9. Fazziki, A.E., Lakhrissi, H., Yetognon, K., Sadgal, M.: A service oriented information system: a model driven approach. In: Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on. pp. 466–473. IEEE (2012)
10. Juve, G., Deelman, E.: Automating application deployment in infrastructure clouds. In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. pp. 658–665. IEEE (2011)
11. Konstantinou, A.V., Eilam, T., Kalantar, M., Totok, A.A., Arnold, W., Snible, E.: An architecture for virtual solution composition and deployment in infrastructure clouds. In: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing. pp. 9–18. ACM (2009)
12. Li, W., Svard, P., Tordsson, J., Elmroth, E.: A general approach to service deployment in cloud environments. In: Cloud and Green Computing (CGC), 2012 Second International Conference on. pp. 17–24. IEEE (2012)
13. Muthunagai, S., Karthic, C., Sujatha, S.: Efficient access of cloud resources through virtualization techniques. In: Recent Trends In Information Technology (ICRTIT), 2012 International Conference on. pp. 174–178. IEEE (2012)
14. Nielsen, J.: Usability engineering. Elsevier (1994)
15. OMG: Uml 2.4.1. http://www.omg.org/spec/UML/2.4.1/, (Visited on 01/06/2016)
16. Salapura, V.: Cloud computing: Virtualization and resiliency for data center computing. In: Computer Design (ICCD), 2012 IEEE 30th International Conference on. pp. 1–2. IEEE (2012)
17. Santos, R.C.: Revisão das métricas para avaliação de usabilidade de sistemas (review of the metrics for evaluating the usability of systems). In: Congresso Internacional GBATA (2008)
18. Savu, L.: Cloud computing: Deployment models, delivery models, risks and research challenges. In: 2011 International Conference on Computer and Management (CAMAN) (2011)
19. Talwar, V., Milojicic, D., Wu, Q., Pu, C., Yan, W., Jung, G.P.: Approaches for service deployment. Internet Computing, IEEE 9(2), 70–80 (2005)
20. Zhang, Y., Li, Y., Zheng, W.: Automatic software deployment using user-level virtualization for cloud-computing. Future Generation Computer Systems 29(1), 323–329 (2013)