Journal of Computers Vol. 29 No. 6, 2018, pp. 230-234
doi:10.3966/199115992018122906022

A Smart Provisioning Approach to Cloud Infrastructure

Julio Sandobalin[1,2*], Emilio Insfran[2], Silvia Abrahao[2]

[1] Departamento de Informática y Ciencias de la Computación, Escuela Politécnica Nacional, Ladrón de Guevara, E11-253, Quito, Ecuador
julio.sandobalin@epn.edu.ec

[2] Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Camino de Vera, s/n, 46022, Valencia, España
{jsandobalin, einsfran, sabrahao}@dsic.upv.es

Received 9 January 2018; Revised 21 January 2018; Accepted 22 January 2018

Abstract. Cloud Computing has become the primary model to pay-per-use used by practitioners and researchers for getting an infrastructure in a short time. DevOps is a paradigm that provides practices and tools to optimize the software delivery time. On the one hand, Infrastructure as Code is the cornerstone of DevOps to infrastructure automation based on practices of software development. On the other hand, there exist tools that provide support for infrastructure provisioning and configuration management in the Cloud. Currently, companies are using Cloud-based DevOps processes to leverage the capacities of Cloud Computing and improve the software delivery time. However, infrastructure provisioning is a time-consuming and error-prone activity because it is supported by tools which work in isolation and they do not have any link between them. For this reason, we propose a smart provisioning approach of infrastructure based on models, which have configuration needed to create infrastructure, run a set of automated tests and setup automatically the DevOps tools used in this process.

Keywords: cloud computing, DevOps, infrastructure provisioning, model-driven development, smart provisioning

1 Introduction

In today's enterprises, one of the most important challenges is how to deliver a new idea or software artefact to customers as quickly as possible. Furthermore, to succeed in a world where technologies, requirements, ideas, tools, and timelines are continually changing, information must be accurate, readily available, easily found and, ideally, delivered continuously in real-time to all team members [1]. For these reasons, practitioners have adopted a new paradigm called DevOps [2] (Development & Operations), which is promoting continuous collaboration between developers and operation staff through a set of principles, practices and tools that optimize the software delivery time. The cornerstone of DevOps is the Infrastructure as Code [3] which is an approach to infrastructure automation based on software development practices that emphasize the use of consistent and repeatable routines for hardware provisioning. We take advantage the concepts aforementioned to develop an infrastructure modelling tool called ARGON [4], which models a generic infrastructure model and generate configuration files to manage provisioning tools. On the other hand, we have presented a DevOps toolchain for cloud infrastructure provisioning [5], which supports an approach to achieve a continuous delivery process for cloud infrastructure provisioning based on DevOps community tools.

In this work, we present an approach to achieve a smart provisioning of cloud infrastructure based on models. We have abstracted the configuration features of DevOps community tools used in an infrastructure provisioning process for adding new modeling elements in ARGON tool. Firstly, it is necessary to models an infrastructure model that define the cloud elements and provisioning tools

* Corresponding Author

configurations. Once the infrastructure model is ready, it is pushed towards a control version system to retain and provide access to every version of every infrastructure model that has ever been stored on it. Next, an integration server takes the infrastructure model from control version system and automatically generate configuration files for a specific provisioning tool depending on what cloud provider is required. Then, a set of automated tests are run against this configuration files to validate their structure and syntaxes. Once configuration files have overcome the set of tests, the integration server uses these scripts and decide which tool provisioning use to deploy the infrastructure toward a particular cloud provider. Finally, infrastructure has been deployed in the Cloud, and a set of automated tests validate the installed hardware and software.

The remainder of this paper is structured as follows: Section 2 discusses related works. Section 3 presents the smart provisioning of cloud infrastructure. Finally, Section 4 presents our conclusions and future work.

2 Related Works

Currently, practitioners and researchers have focused their efforts on cloud infrastructure provisioning using DevOps community tools. In this context, below are described the principal research works that aim this approach.

MODAClouds [6] is a European project undertaken to simplify the management of cloud service. The goal is to deliver an Integrated Development Environment to support system developers in building and deploying applications and related data to multi-Clouds spanning across the full Cloud stack. Energizer 4Clouds is an executable platform of MODAClouds which includes automatic infrastructure provisioning using specially designed Puppet modules, the ability to use existing infrastructure and an API middleware for job control.

MORE [7] is a model-driven operation service for cloud-based IT systems that focus on automating the initial deployment and the dynamic configuration of a system. MORE provides an online modelling environment for defining a topology model to specify system structure and desired state. MORE transforms the topology model into executable code for the Puppet tool to get virtual machines, physical machines, and containers.

TOSCA [8] is a standard for Topology and Orchestration Specification for Cloud Application which allows modelling nodes and orchestrates the deployment of cloud applications. TOSCA uses DevOps provisioning tools such as Chef for infrastructure provisioning and Juju for implementation of cloud-based applications.

An analysis of works mentioned above shows that research efforts have focused on reuse DevOps provisioning tools, which are used in isolation and they do not have links for supporting a continuous delivery process of the infrastructure provisioning. For this reason, we propose an approach to support a smart provisioning of cloud infrastructure based on models that use DevOps community tools.

3 Smart Provisioning of Cloud Infrastructure

There is a wide range of cloud providers and tools that can be used to support the infrastructure provisioning and software applications deployment. To support these processes we have presented an infrastructure modeling tool for cloud provisioning called ARGON [4], which was developed following the Model Driven Architecture (MDA).

ARGON has a Domain Specific Language (DSL) to model the cloud infrastructure and a transformation-engine to generate configuration files automatically. ARGON has an Infrastructure Metamodel (see Fig. 1), which abstracts the capacities of cloud computing (i.e. computing, storage, networking and elasticity) and the configuration files for a specific cloud provider. The Infrastructure Metamodel is the abstract syntax used to generate the graphical notation or concrete syntax of the DSL. In the Infrastructure Model (see Fig. 2) are modeled generic cloud elements and configurations, which will be used at runtime to decide between available provisioning tools that are useful for a particular cloud provider.

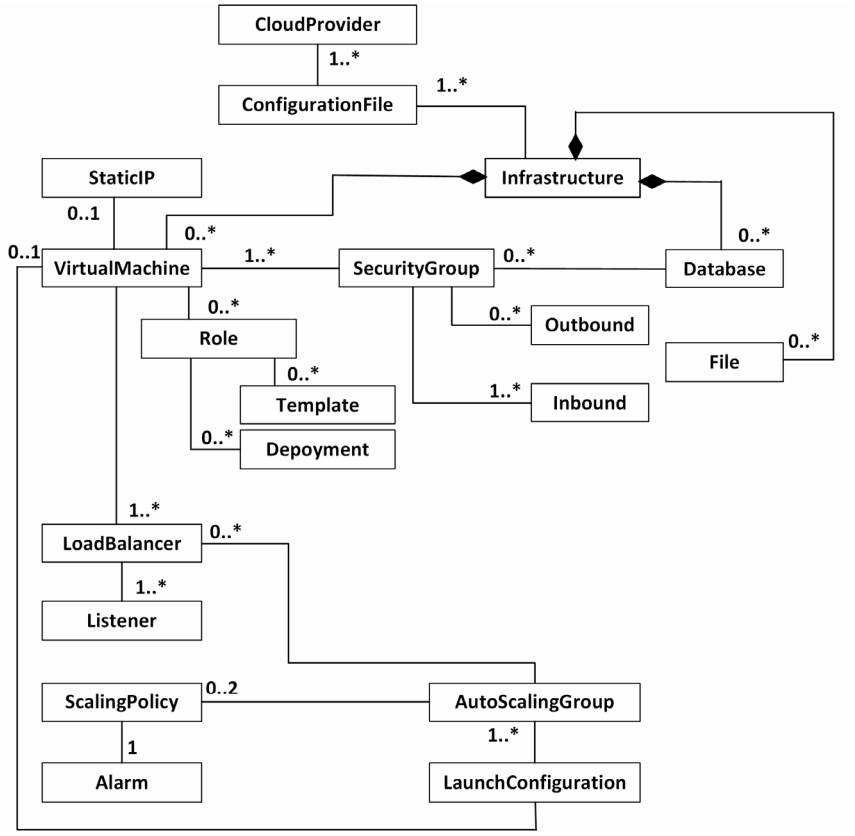

Fig. 1. Infrastructure Metamodel

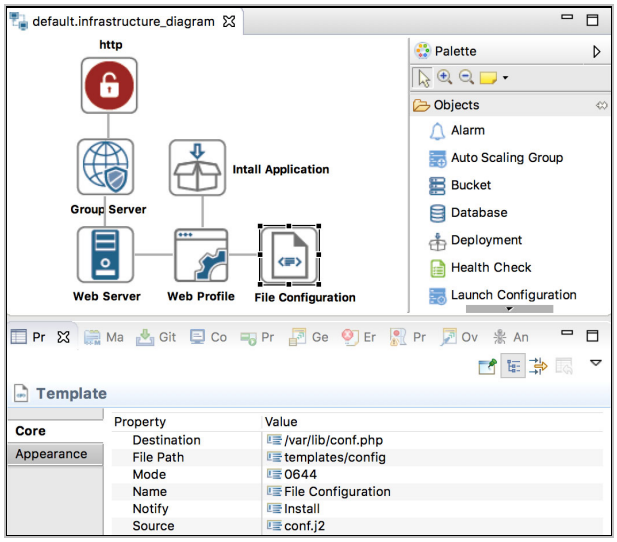

Fig. 2. An Infrastructure Modeling Too

ARGON architecture (see Fig. 3) uses the Infrastructure Metamodel as a Platform-Independent Model (PIM) to facilitate the model-to-model (M2M) transformation and get a specific Infrastructure Model as a Platform-Specific Model (PSM) and settings to choose a provisioning tool. Once the particular provisioning tool and cloud provider are selected, the transformation-engine uses the Infrastructure Model to apply on it a set of model-to-text (M2T) transformation and generate scripts, which have the sentences to create infrastructure in the Cloud.

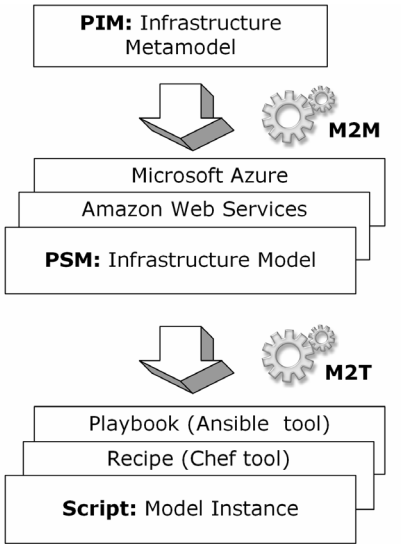

Fig. 3. ARGON architecture

ARGON has a Domain Specific Language (see Fig. 3) to model the cloud infrastructure elements and its configuration platforms.

In the Fig. 3, it is depicted a cloud application in which a virtual machine (***Web Server***) has a security group (***Group Sever***), which allows the access to the virtual machine through the port 80 (***http***). The virtual machine has a role (***Web Profile***), which specified the software application (***Install Application***) that will be installed and configurations (***File Configuration***). Furthermore, it will be stored in a knowledge database and used by configuration file repository to choose a provisioning tool for deploying the cloud infrastructure.

The attributes tab of File configuration (see Fig. 3) is used by Configuration File Repository to give an intelligent provisioning to the cloud infrastructure. In these attributes are specified:

- ***destination*** and ***file path*** specify configuration files that will be modified in the virtual machine deployed,
- ***mode*** defines the file permission which must be applied in the file configuration,
- ***name*** is the element name on the infrastructure model,
- ***notify*** is an alarm that is activated in the install process, and
- ***source*** is a configuration file with the setting of provisioning tool and cloud provider.

Once the cloud infrastructure provisioning and its configuration files are modelled in ARGON, it starts the smart provisioning pipeline of cloud infrastructure (see Fig. 4). The infrastructure model (see Fig. 3) is pushed toward a version control system to retain and provide access to every version of every infrastructure model that has ever been stored on it. An integration server like Jenkins or Hudson take this infrastructure model and start the automated infrastructure provisioning. The transformation-engine installed in the integration server apply over the infrastructure model a set of transformation rule to get scripts with the instructions needed to create infrastructure in the Cloud. Moreover, configuration files to set the provisioning tools like Ansible or Puppet are generated. Both scripts and configuration files have to overcome a set of automated tests to validate its structure and syntax.

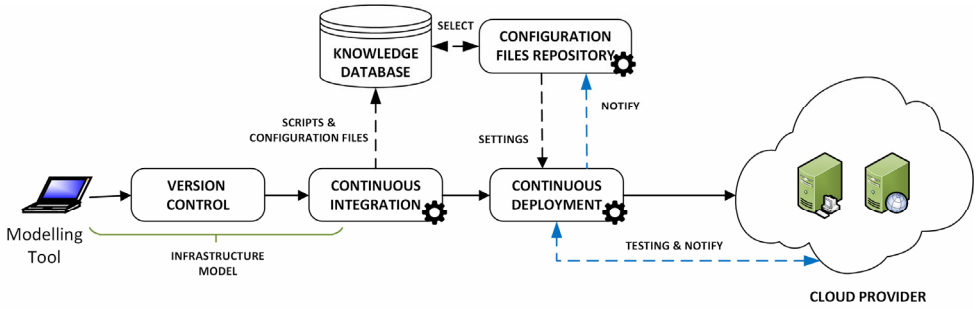

Fig. 4. Overview of an intelligent provisioning pipeline of cloud infrastructure

The configuration files are stored in a knowledge database and, subsequently, in a configuration file repository is run a set of basic decision rules to select a configuration file to depend on both the infrastructure provisioning tool and the cloud provider, which are specified in the infrastructure model. In the case of the infrastructure provisioning process failed, the configuration file repository will select another configuration file to repeat the provisioning process until it works successfully. Therefore, configuration files are used for setting the provisioning tools and cloud providers. Moreover, scripts are used in infrastructure provisioning tools to deploy the hardware and software in the cloud provider.

Finally, over the infrastructure deployed and software installed a set of tests are run against them to verify if the infrastructure was created successfully and the software was installed.

4 Conclusions

In this work, we have presented an approach to achieve a smart provisioning of cloud infrastructure based on models. We take advantage of the information modelled in an infrastructure model to generate scripts for provisioning infrastructure and configuration files to set provisioning tools and cloud providers. We use a knowledge database to store all scripts and files configuration created and, then, configuration file repository should use them to select a file to be used. We call this process like intelligent provisioning because, in the case of a failure in the provisioning process, a new file configuration is selected to repeat the process until it works successfully. With this approach, we tried to reach a continuous delivery process of cloud resources based on an intelligent provisioning of cloud infrastructure and DevOps. The scope of this proposal provides a smart in the infrastructure provisioning stage.

As a future work, we want to extend the concept of smart provisioning and provide self-learning through files configuration and scripts to give a holistic solution to the of cloud infrastructure provisioning.

Acknowledgements

This research is supported by the Value@Cloud project (TIN2013- 46300-R).

References

[1] C.A. Cois, J. Yankel, A. Connell, Modern DevOps: Optimizing software development through effective system interactions, in: Proc. IEEE International Professional Communication Conference, 2015.

[2] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1st ed., Addison-Wesley Professional, Boston, MA, 2010.

[3] K. Morris, Infrastructure as Code: Managing Servers in the Cloud, 1st ed., O'Reilly Media, Sebastopol, CA, 2016.

[4] J. Sandobalin, E. Insfran, S. Abrahao, An infrastructure modelling tool for cloud provisioning, in: Proc. 14th IEEE International Conference on Services Computing, SCC, 2017.

[5] J. Sandobalin, E. Insfran, S. Abrahao, End-to-end automation in cloud infrastructure provisioning, in: Proc. 26th International Conference on Information Systems Development, ISD, 2017.

[6] E. Di Nitto, P. Matthews, D. Petcu, A. Solberg, Model-Driven Development and Operation of Multi-Cloud Applications, Springer International Publishing, Cham, 2017.

[7] W. Chen, C. Liang, Y. Wan, C. Gao, G. Wu, J. Wei, T. Huang, MORE: a model-driven operation service for cloud-based IT systems, in: Proc. 13th IEEE International Conference on Services Computing, SCC, 2016.

[8] J. Wettinger, U. Breitenbücher, O. Kopp, F. Leymann, Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel, Future Generation Computer Systems 56(C)(2016) 317-332.