C2C: An Automated Deployment Framework for Distributed Applications on Multi-Clouds

Flora Karniavoura, Antonis Papaioannou, and Kostas Magoutis

Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
Heraklion 70013, Greece
{karniav,papaioan,magoutis}@ics.forth.gr

Abstract. The Cloud Application Modeling and Execution Language (CAMEL) is a new domain-specific modeling language (DSL) targeted to modeling applications and to supporting their lifecycle management on multiple (heterogenous) cloud providers. Configuration management tools that provide automated solutions to application configuration and deployment, such as Opscode Chef, have recently met wide acceptance by the development and operations (or DevOps) community. In this paper, we describe a methodology to map CAMEL models of distributed applications to Chef concepts for configuration and deployment on multi-clouds. We introduce C2C, a tool that aims to automate this process and discuss the challenges raised along the way suggesting possible solutions.

Keywords: Cloud computing, Application modeling, Configuration management

1 Introduction

In the era of cloud computing, applications are benefitting from a virtually inexhaustible supply of resources, a flexible economic model, and a rich choice of available providers. Applications consisting of several software components or services typically need to be deployed over multiple underlying technologies, often across different cloud providers. To bridge across different cloud environments, tools based on model-driven engineering principles are recently gaining ground among developers and operations engineers. TOSCA [1], CloudML [5] and CAMEL [17] are three recently introduced model-driven approaches used to express application structures and requirements, and to manage application deployments over time.

An important tool in the hands of application developers and operations engineers is the ability to maintain a detailed recording of software and hardware components and their interdependencies in an infrastructure, in a process known as *configuration management* (CM) [13]. An effective CM process provides significant benefits including reduced complexity through abstraction, greater flexibility, faster machine deployment and disaster recovery, etc. There are numerous configuration management tools from which the most widely known are:

Bcfg2 [3], CFEngine [6], Chef [7], and Puppet [16]. Each of these tools has its strengths and weaknesses [19] [9]. A CM solution is often combined with provisioning and deployment tools.

In this position paper we bridge the gap between application models (which are typically declarative expressions of application state) and configuration management tools (imperative procedures for carrying out CM actions) using CAMEL and Chef as specific instances of the two approaches. We introduce *CAMEL-to-Chef* (or C2C for short), a new methodology for the deployment and configuration of applications expressed in CAMEL across multi-cloud environments.

2 Background

2.1 CAMEL

Cloud Application Modeling and Execution Language (CAMEL) is a family of domain-specific languages (DSLs) currently under development in the PaaSage EU project [14]. CAMEL encompasses DSLs covering a wealth of aspects of specification and execution of multi-cloud applications. CloudML, one of the DSLs comprising CAMEL, is used to describe the application structure and specify the topology of virtual machines and applications components. Below we describe key modeling elements that CAMEL shares with CloudML.

- *Cloud*: a collection of virtual machines (VMs) offered by a cloud provider
- *VM type*, *VM instance* : a VM type refers to a generic description of a VM, while an instance of a VM type refers to a specific instantiation of a VM, including specific configuration information.
- *Internal component* : a reusable type of application component, whereas an *internal component instance* represents an instance of an application component. The description of an application component stays at a generic level while the specification of its respective instances involves particular configuration information.
- *Hosting*, *Hosting Instance* : a hosting relationship between a host VM and a component of the application, or between two application components.
- *Communication*, *Communication Instance* : a dependency relationship between two application components or component instances.

CAMEL is under development at this time and thus constantly evolving. The changes that have been brought into CAMEL since we started the C2C project have so far been dealt with with just minor changes at the model parsing phase and have not resulted in drastic changes in the fundamentals of our approach.

2.2 Opscode Chef

Chef is a configuration management tool created by Opscode [7]. Following an infrastructure-as-code approach, Chef uses *Recipes*, configuration files written in Ruby that describe the actions that should be performed on a node in order

to bring it to its desired state. Related recipes are stored in *Cookbooks*. Users can store and write Cookbooks at the Chef repository in their local workstation, from where they can also interact with the Chef server. Every machine-node that is managed by Chef has a *run-list*, which is the list of recipes that will run on it at the time of the next Chef client run. We should note that dependencies between cookbooks are handled automatically by the Chef server, which is also responsible for various other tasks like run-list and cookbook storing.

Chef brings in a number of benefits. It offers automated and reusable solutions for the configuration and deployment of applications and a lot of ready-to-use, publicly available Cookbooks via the Chef repository, also known as *Chef supermarket* [8]. One of the strongest aspects of Chef is its active and constantly evolving community. The Chef community consists of people of various backgrounds and expertise that contributes to the creation and improvement of a large set of Cookbooks covering a wide range of software components.

3 Related work

Application modeling is becoming increasingly popular nowadays due to the complexity and increased needs of distributed applications. A recently introduced modeling approach covering the description, deployment, and lifecycle management of distributed applications is TOSCA [1]. Perhaps closest to our approach is a recent paper on cloud service orchestration using TOSCA, Chef and openstack [11] uses Chef as a deployment tool for applications defined as TOSCA models. "Deployment artifacts" are defined at the time of model creation for each component, stating which Cookbook recipe(s) should be used to deploy them. Deployments take place on openstack and various Chef functions are triggered using the knife-openstack client [12]. The differences between this work and ours are (1) the fact that we use CAMEL instead of TOSCA to model our applications, and (2) we automatically derive information from CAMEL models to achieve deployment with Chef in *multi-cloud* environments. The CAMEL model does not need to contain information about the recipes needed for each component, although we describe this as an alternative technique in Section 6.

CloudML [5] also offers a deployment and lifecycle management mechanism [4] by associating each deployable component with a pointer code responsible for its deployment. The deployment process is restricted to scripted commands and does not envolve the usage of Chef.

4 The C2C methodology

4.1 Architecture

Figure 1 depicts the overall architecture of our system. C2C comprises three major modules: i) the model parser ii) the VM manager and iii) the Chef instructor. The *model parser* analyses the application model given as input, extracts the

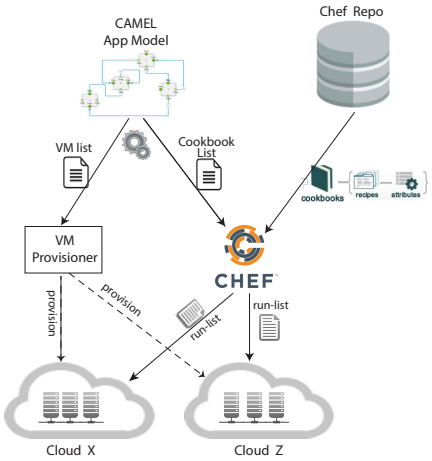

Fig. 1. System architecture

necessary information and prepares the input for the other C2C modules. In more details it forms a list containing the VMs that will be used for the application deployment. In addition it prepares the input of the Chef instructor module which is a list containing the software components that comprise the application along with their hosting and communication relationships. The *VM manager* module is responsible for the provisioning of the VMs and install the Chef client in each one. The *Chef instructor* is responsible for the deployment of the application software components on the appropriate VM, indicated by the hosting instances. As a first step, it collects all the necessary Cookbooks by searching at the Chef workstation or on the Chef's community repository [8]. Next it forms the run-list of each node in order to install the application components. The Chef instructor derives the order in which the components should be installed as well as the node that will host each one by analysing the communication and hosting relationships among the components.

We follow this modular approach because it allows us to split the functionality of our tool and minimize the amount of effort needed in case of a component update (e.g., in case of a newer version of CAMEL we have to update only the model parser, or in case we want to support more Cloud providers we should update the VM manager component).

We implemented the model parser using the Java compatible CAMEL API library. The multi-Cloud provisioning logic we embedded in VM manager uses the third party library of *Apache JClouds* [10] and the official *Azure Java sdk* [2] API to operate across the different Cloud architectures of Openstack, Flexiant, Amazon EC2 and Microsoft Azure.

4.2 Mapping of concepts

In this section we describe the necessary CAMEL attributes that are used by the model parser in order to prepare the input of the other C2C modules. The *VM instance* properties contain all the necessary information for the VM manager to provision the required resources for the application deployment. The Chef instructor uses the names of the *Component instances* in order to identify the corresponding cookbooks. The *hosting instance* relationship between a (software) component instance and an VM instance indicates that the corresponding cookbook should be added to the run-list of the node. On the other hand if a component instnace A is hosted in component instance B, then the cookbook corresponding to B should also be included in the run-list of node that will host component A. The deployment order of the components is derived based on the the *hosting instances* and the *communication instance* between component instances. For example if component X communicates with component Y then the deployment of Y should precced the deployment of X.

5 Use case

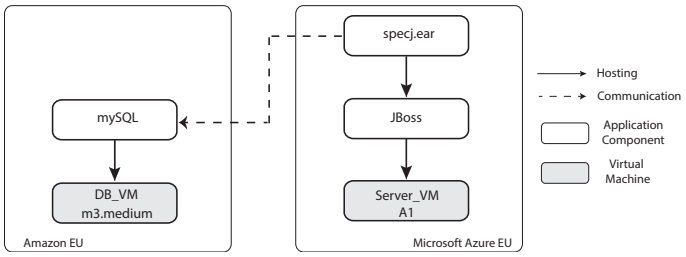

Fig. 2. Spec jEnterprise2010 application structure

We demostrate our systems functionality using the distributed SPEC jEnterprise2010 benchmark [18] as a case study. SPEC jEnterprise2010 is a full system benchmark that allows performance measurement of Java EE servers and supporting infrastructure. The SPEC jEnterprise2010 application requires a Java EE application server and a relational database management system.

We model the SPEC jEnterprise2010 application using three software components corresponding to the business logic of the application, the application server and the RDBMS [15]. These components are instantiated as a specj.ear, a JBoss application server and a MySQL database. Figure 2 presents the application structure and the deployment scenario of SPEC jEnterprise2010. The solid line arrows indicate the hosting relationships between VMs and application components as well as the communication among software components. The dashed line arrows represent the communications between the application components.

In this scenario we demonstrate a cross-Cloud deployment of the application (different application components are deployed on different Cloud platforms).

At the first step the model parser instructs the VM manager to provision a m3.medium VM on Amazon EC2 platform and an A1 VM on Azure. Then the Chef instructor fetches the necessary cookbooks of MySQL and JBoss from the Chef supermarket. On the other hand we provide our custom cookbook for the deployment of the application logic (specj.ear) in our local workstation. The hosting and communication relationships between software components and VMs suggest the run-lists of Chef nodes. The run-list corresponding to the DB_VM contains the cookbook of MySQL while the run-list of Server_VM contains the cookbooks of and specj.ear. The deployment of DB node precedes the server node according to the Chef instructor logic described in section 4.

6 Challenges

Our aim for the C2C methodology is to be as automatic as possible, however there are challenges to achieve this that we discuss in this section along with possible solutions.

A key challenge in the C2C methodology is to decide automatically what is the correct Chef cookbook to use for a particular software component. In our automatic implementation we assume that there is a match between a component's name and the name of the suitable cookbook for it – however this need not always be true.

If we assume that components in CAMEL models are named using some variation of the name of the software component that they model, C2C could map them to cookbooks whose names most closely match the name of the software component (e.g., exhibit minimal lexicographical distance from it). This solution could be error-proof if the creator of a CAMEL model is aware of the Chef cookbook it wants to map the component to and thus names the component using the exact name of the Chef cookbook.

However even if the right cookbook for a component is discovered, the problem has not been solved. The difficulty now lies in distinguishing the right recipe for the desired task, between all recipes in the cookbook. In most cases the "default" recipe, present in all cookbooks, is responsible for the basic cookbook task (in most cases, installation) but this does not apply to every cookbook. Usually, recipe names are quite descriptive but not to an extent that could lead to efficient recipe selection.

A solution to this problem could be an appropriate naming scheme for cookbook recipes. Firstly, unique keywords such as "install", "update" or "start" should be used for basic tasks implemented by recipes. Cookbook recipes could simply include annotations stating which of these tasks each one of them implements. Current recipes do not provide this kind of information but it could be retrofitted or overlaid on recipe metadata based on user feedback: cookbook users could report on the task each recipe they use performs, and this information could later be used to help C2C automatically choose the right recipe.

Finally, a simple way to address these issues is by declaring the exact cookbook recipes to use in each application component within the CAMEL model (similar to what was proposed in [11]). This solution eliminates the risk of choosing the wrong recipe, albeit at the cost of reduced flexibility.

7 Conclusion

In this position paper we introduced C2C, an automated deployment framework for distributed applications on multi-clouds. We showed that the configuration, deployment and lifecycle management of CAMEL applications leveraging the large base of Chef cookbooks is achievable in an automated fashion. We discussed the challenges that stand in the way of full automation with this methodology and proposed different ways to overcome them. Finally, we demonstrated the usability of C2C using the SPEC jEnterprise2010 application as a case study.

References

1. Oasis: Oasis topology and orchestration specification for cloud applications (tosca)
2. Azure SDK (Accessed 1/2015), `https://github.com/Azure/azure-sdk-for-java`
3. Bcfg2: (Accessed 2/2015), `http://www.bcfg2.org/`
4. Blair, G., Bencomo, N., France, R.: Models@ run.time. Computer 42(10) (2009)
5. Brandtzg, E., Parastoo, M., Mosser, S.: Towards a Domain-Specific Language to Deploy Applications in the Clouds. In: CLOUD COMPUTING 2012: 3rd International Conference on Cloud Computing, GRIDs, and Virtualization (2012)
6. CFEngine: (Accessed 2/2015), `http://www.cfengine.com/`
7. Chef: (Accessed 1/2015), `http://www.getchef.com/`
8. Chef Supermarket: (Accessed 1/2015), `https://supermarket.chef.io/`
9. Delaet, T., Joosen, W., Vanbrabant, B.: A survey of system configuration tools. In: Proceedings of the 24th International Conference on Large Installation System Administration. pp. 1–8. LISA'10 (2010)
10. JClouds: (Accessed 1/2015), `https://jclouds.apache.org/`
11. Katsaros, Menzel, L.: Cloud service orchestration with tosca, chef and openstack pp. 1–8 (2014)
12. Knife-Openstack client: (Accessed 2/2015), `https://docs.chef.io/plugin_knife_openstack.html`
13. Lueninghoener, C.: Getting started with configuration management. ;login: 36(2), 12–17 (2011)
14. PaaSage EU FP7 project: (Accessed 2/2015), `http://www.paasage.eu/`
15. Papaioannou, A., Magoutis, K.: An architecture for evaluating distributed application deployments in multi-clouds. In: Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on. vol. 1 (Dec 2013)
16. Puppet: (Accessed 2/2015), `http://www.puppetlabs.com/`
17. Rossini, A., Nikolov, N., Romero, D., Domaschka, J., Kritikos, K., T., K., Solberg, A.: Paasage project deliverable d2.1.2 - cloudml implementation documentation. Public deliverable (2014)
18. SPEC jEnterprise2010: (Accessed 2/2015), `https://www.spec.org/jEnterprise2010/`
19. Tsalolikhin, A.: Configuration management summit. ;login: 35(5), 104–105 (2010)