Continous Deployment of Multi-cloud Systems

Nicolas Ferry, Franck Chauvel, Hui Song, Arnor Solberg
Department of Networked Systems and Services, SINTEF, Oslo, Norway
{name.surname}@sintef.no

ABSTRACT

In this paper we present our mechanism and tooling for the continuous deployment and resource provisioning of multi-cloud applications. In order to facilitate collaboration between development and operation teams as promoted in the DevOps movement, our deployment and resource provisioning engine is based on the Models@Runtime principles. This enables applying the same concepts and language (*i.e.*, CLOUDML) for deployment and resource provisioning at development-and operation-time.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Configuration Management

Keywords

Cloud computing, deployment, model-driven engineering, Models@Runtime, CloudML

1. INTRODUCTION

Multi-cloud systems are typically large scale, distributed, and dynamic. The need to evolve and update such systems after delivery is often inevitable, for example due to: changes in the requirements, needs for maintenance, or demands for advancing the quality of service such as scalability and performance. In order to ensure the quality of the system, validation and analysis tasks have to be carried out during all development steps towards the delivery of the system. This includes testing the deployment as well as the various features of the system in a pre-production environment. Such testing usually involves both software developers that are responsible for designing and implementing the system according to the functional and non-functional requirements, and software system operators that are responsible for operating the system at runtime.

Traditionally, these teams are working separately, typically using specific languages and tools they feel comfortable with [3]. In addition, languages and tools typically vary within these teams since various cloud platforms offer different languages and tools [3]. This can hinder knowledge sharing, making it difficult: i) for designers to obtain and understand feedback on the status of the operated system that would be useful in order to evolve the system, ii) for operators to analyse and comment on the impact of proposed or implemented changes to the system, and iii) for operators of different cloud platforms to exchange knowledge and specific operation management decisions. The DevOps movement [4] promotes to facilitate collaboration between developers and operators, for example, through aligning concepts and languages used in development and operation and supporting them with automated tools that help reducing the gap and improving the flexibility and efficiency of the delivery life-cycle.

In this tool paper we present the CLOUDMF mechanism and tooling to reduce the gap between development and operation teams by supporting continuous deployment of multi-cloud applications. In order to reduce this gap, we apply the same concepts and language at development-time and at operation-time. To automate the continous deployment and resource provisioning, we have developed an engine based on the principles of the Models@Runtime approach [1]. The deployment engine "speaks" the language of CLOUDML [2], thereby, it enables to apply the same concepts and abstractions for the operators as applied by the developers. Moreover, it enables seamless operation of multi-cloud deployment and provisioning since our previously developed CLOUDML modelling language provides abstractions enabling to specify deployment and resource provisioning in a cloud platform agnostic way.

2. CONTINUOUS DEPLOYMENT USING MODELS@RUNTIME

Models@Runtime [1] is an architectural pattern for dynamic adaptive systems that leverage models as executable artefacts that can be applied to support the execution of the system. Thus, Models@Runtime can be applied to reduce the developer-operator gap by providing a unique model-based representation of the applications that can be applied for both design- and run-time activities. As depicted in Figure 1, Models@Runtime enables to provide abstract representations of the underlying running system, which facilitates reasoning, analysis, simulation, and adaptation. A change in the running system is automatically reflected in the model of the current system. Similarly, a modification to this model is enacted on the running system on demand. This causal connection enables the continuous evolution of the system with no strict boundaries between design-time and run-time activities.

In our tooling we exploit Models@Runtime for the continuous deployment of cloud-based applications following the process depicted in Figure 1. A developer team specifies a model of the deployment and provisioning of its application applying CLOUDML, and then automatically enacts this deployment and provisioning in a test environment. Developers exploit the test environment to

QUDOS'15, September 1, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3817-2/15/09...$15.00
http://dx.doi.org/10.1145/2804371.2804377

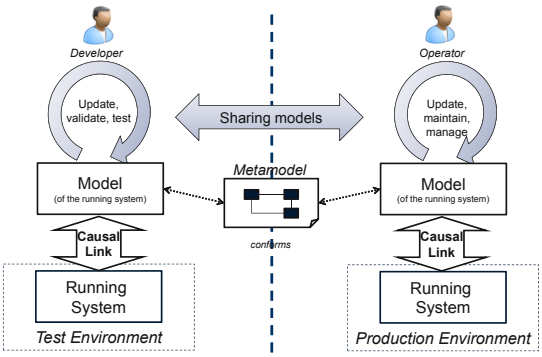

Figure 1: Continuous deployment using Models@Runtime

check, analyse and validate the deployment and resource provisioning as well as the system's features, and continously tune its development and redeploy it automatically. Any change made to the deployment model will be enacted on demand on the running system whilst its status will be reflected in the model providing useful feedbacks. Once the new release is validated, it can be provided together with the associated CLOUDML model to the operation team. The latter can in turn exploit the model to deploy the new release in a production environment. The operators can further tune this model to maintain, manage and evolve the running system. Because the models shared by the developers and operators conform to the same metamodel, at any time they can share and exchange information. Moreover, any automation of the ''hand-over'' from the Test Environment to the Operation Environment can be provided seamlessly.

3. THE DEPLOYMENT ENGINE

Figure 2 depicts the architecture of our Models@Runtime based deployment engine. A reasoning engine (*e.g.*, operator, analysis engine) can read the current model provided by the deployment engine (step 1), which describes the actual running system, and then produces a target model (step 2). Then, the deployment engine calculates the difference between the current and the target model (step 3). Finally, the deployment engine enacts the changes modifying only the parts of the system necessary to account for the difference, and the target model becomes the current (step 4).

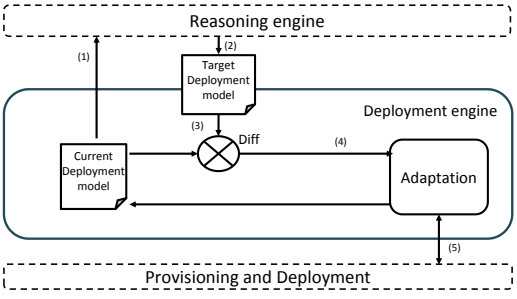

Figure 2: The Models@Runtime based Deployment Engine

Figure 3 shows the web interface for viewing and manipulating the deployment and resource provisioning either in the test environment at design-time or in the operation environment at run-time.

Adaptations in the deployment of a cloud-based application can be specified by providing a new deployment model describing the

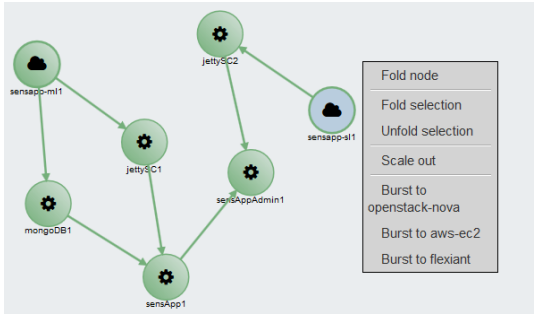

Figure 3: Snapshot of the CLOUDMF Web-based editor

desired deployment. A Comparison engine compares the current and target models and derives a plan describing how to reach that deployment. The resulting plan modifies only the parts of the system necessary to account for the difference thus minimizing the impact on the system in production (from a quality point it is important to not touch the running system more than necessary).

The comparison engine processes the entities composing the deployment models on the basis of their logical dependencies. In this way, all the components required by another component are deployed first. For each of these components, the engine compares the two sets of instances from the current and target models. This comparison is achieved on the matching of both the properties of the instances and their types as well as on the basis of their dependencies (*e.g.*, if the host of a software component has changed it might be redeployed). For each unmatched instance from the current model a `remove` action with the instance as argument is created. Similarly, for each unmatched instance from the target model an `add` action with the instance as argument is generated. In addition, by analysing the difference between the two models, the comparison engine identifies the high level operations to be performed. For instance, when bursting to a new provider, the engine triggers a classical deployment, whilst when scaling within the same cloud, it creates and provisions image of the VM to be scaled and then reconfigures and restarts the components hosted on it.

4. CONCLUSION

Our Models@Runtime approach leverages upon MDE techniques and methods at runtime to support the continuous design and deployment of multi-cloud applications. Thanks to the proposed approach, it becomes possible to exploit the same concepts and language for deployment and resource provisioning at both development and operation time.

Acknowledgements. The research leading to these results has received funding from the European Community's FP7 program under grant agreement number: 318484 (MODAClouds).

5. REFERENCES

[1] G. Blair, N. Bencomo, and R. France. Models@run.time. *IEEE Computer*, 42(10):22–27, 2009.

[2] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg. CloudMF: Applying MDE to Tame the Complexity of Managing Multi-Cloud Applications. In *UCC*, 2014.

[3] M. Httermann. *DevOps for developers*. Apress, 2012.

[4] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.