Will Be Defined After the Brainstorming Phase*

Eirik Brandztæg[1,2] and Sébastien Mosser[1]

[1] SINTEF IKT, Oslo, Norway
[2] University of Oslo, Oslo, Norway
{firstname.lastname}@sintef.no

Built: February 7, 2012

Abstract. ∼150 words expected. Will also be defined after the brainstorming phase. Must mention *(i)*the problem, *(ii)*the actual contribution and *(iii)*the obtained results.

1 Introduction

I'll write the introduction afterwards, when the content of the paper will be fixed.

- Cloud–computing research field [3]
- Model–driven engineering applied to the cloud

2 Challenges in the cloud

To recognize cloud provisioning challenges an example application [4] has been utilized. The application (from here known as BankManager) is a featureless bank manager system written in Grails [8], it supports creating users and bank accounts, moving money between bank accounts and users. BankManager is designed to be distributed between several nodes with two front-end applications connected to one back-end database as seen in Figure 1 with three nodes and one browser to visualize application flow. A typical enterprise scenario for a modern business. To prototype deployment of the design scripts were used to create instances and deploy the software. From this prototype it became clear that there were several challenges that CloudML must tackle:

- **Dependent on data**: Even though provisioning is the main focus of this article it proved complex and difficult to deploy BankManager without runtime information from the provisioning step. We will use models-at-runtime **source, what is models-at-runtime?** to solve this problem
- **Technical competence**: The level of knowledge needed to manually provision nodes is more technical and not business liable. Even more so for multicloud provisioning

* This work is funded by the European commission through the REMICS project, contract number 257793, with the 7th Framework Program.

– **Reproducibility**: Once a cluster of nodes are manually initialized it is difficult to replicate the setup on other clouds, or even on the same provider. We will use models-at-runtime to achieve a configuration that can be replicated
– **Robustness**: There were several ways the scripts could fail and most errors were ignored
– **Complexity**: There exists standards [6] for clouds and provisioning on cloud environments. But the providers we tested against [2, 7] did not behave equally, and the APIs were different
– **Shareable**: When we designed the layout of propagation, such as amount of nodes and their specific properties, there were no consistent way to interact structural plans among members of the exercise. Our solution will solve this merely by the languages core design

Use tikz instead?

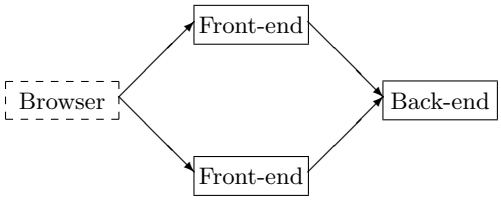

Fig. 1. Nodes for deploying BankManager

3 Contribution

CloudML is a lexical model-based language for cloud provisioning designed to solve the challenges presented in Sec. 2. The language has two main models shown in Fig. 2, *Account* and *Template*. The *Account* model is used for provider credentials to authenticate against a given provider such as Amazon or Rackspace. This model can be stored, shared and reused for the same provider several times. *Template* model on the other hand is used to define sets of resources **source** such as nodes or load balancers, also known as a *stack* **source**. This model can also be stored, shared and reused like the *Account* model, but it does not alternate between providers (*Accounts*), once a template is constructed it can be reused to create the same *stack* on other supported providers. It can even be reused to multiply a setup, without former knowledge of the system as a constraint which directly solved the problem of **reproducibility**. The two models use the same lexical syntax and therefore share the benefit of being **shareable** through mediums such as e-mail and can be properly used on version control systems such as Subversion or Git. The models are separated in

the figure because combining them is semantically wrong, as they are only used together during building without any link to runtime models. Both models are regarded as building blocks in a model-based approach, given their distinct similarities to models described in *Virtual Deployment Model* by Konstantinou [1], only directed against provisioning instead of complete deployment. This model-based provisioning approach is beneficial in regard to the challenge of **technical competence**, since the models can be visualized with graphical models on tools such as whiteboards. This makes it easier for people with less technical knowledge to discuss, edit and even design propagation configuration. The template model of CloudML has a counter model, a model designed specifically for runtime environment. This model-at-runtime **source** provides information about nodes in form of properties, this data is essential to overcome the issue of being **dependant on data** from started nodes when performing cloud deployment.

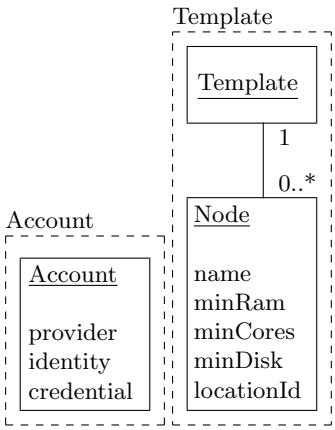

Fig. 2. Model of Account and Template

Implementation. CloudML is implemented as a proof of concept framework [5] (from here known as *cloudml-engine*). Cloudml-engine is written in Scala, builds with Maven and can be used from Java. The templates from the contribution are constructed with JavaScript Object Notation (JSON). Clouml-engine use jclouds.org library to connect with cloud providers, giving it support for several providers out of the box to minimize **complexity** as well as stability and **robustness**.

4 Validation & Experiments

– Three instances
  1. Frontend webapp

2. Frontend webapp
3. Backend database

– Setup works on
  - AWS EC2
  - Rackspace cloudservers

5 Related Works

– AWS CloudFormation (Amazon only)
– jclouds (Only through (more advance?) code)
– libcloud (Only through (more advance?) code)
– CA Applogic (only graphical, and inhouse)

6 Conclusions

I'll write the conclusions afterwards.

References

1. A. V. Konstantinou, T. Eilam, M.K.A.A.T.W.A., E.Snible: An architecture for virtual solution composition and deployment in infrastructure clouds. Tech. rep., IBM Research (2009)
2. Amazon: Amazon web services (2012), http://aws.amazon.com/
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (Feb 2009), http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html
4. Brandtzg, E.: Bank manager (2012), https://github.com/eirikb/grails-bank-example
5. Brandtzg, E.: cloudml-engine (2012), https://github.com/eirikb/cloudml-engine
6. Force, D.M.T.: Open virtualization format (2012), http://www.dmtf.org/standards/ovf
7. Rackspace: Rackspace cloud (2012), http://www.rackspace.com/cloud/
8. SpringSource: Grails (2012), http://grails.org