UNIVERSITY OF OSLO
Department of Informatics

CloudML

A DSL for model-based realization of applications in the cloud

Master thesis

Eirik Brandtzæg

Spring 2012
Built: 15th April 2012

CloudML

Eirik Brandtzæg

Spring 2012
Built: 15th April 2012

Abstract

Contents

List of Figures

List of Tables

Preface

Part I

Introduction

Part II

Contribution

Chapter 1

Vision, concepts and principles

In this chapter the core idea of CloudML will be presented. The concept and principle of CloudML is to be an easier and more reliable path into cloud computing for IT-driven businesses of variable sizes. The tool is visioned to parse and execute template files representing topologies and provision these as instances available in the cloud.

The vision of CloudML is reflected through FIG. **??** which gives an overview of the procedure flow in and around CloudML.

Domain of CloudML. Inside FIG. **??** there is an area pointed out as *CloudML*, this area contain components necessary to implement in order to fulfill the vision as a whole. Every part within the designated area is some physical aspect in the implementation, and therefore core parts of the contribution.

The actors. In FIG. **??** there are three actors, *(i)*business person representing someone with administration- or manager position which defines and controls demands for application functionality and capabilities. The next actor, *(ii)*cloud expert has a greater knowledge of the cloud domain *e.g.*, cloud providers, services these offer, limitations, API support and prices. The last actor, *(iii)*user is a person which directly utilize CloudML to do provisioning. This physical person may or may not have the role of cloud expert, hence the the cloud expert extends from the user actor.

Application and topologies. The business person is in charge of the application, he/she has a need for an application that can fulfill certain tasks, and to handle these tasks application demands are made. The cloud expert use the requirements sketched by the business person to define and design node topologies which tackles the application demands. A topology is a map of nodes connected together in a specific pattern, defined by the cloud expert. In a topology there is also information about node attributes *e.g.*, **ac:CPU!** (**ac:CPU!**) power and **ac:RAM!** (**ac:RAM!**) sizes. He/she might create several topologies to fulfill the application demands.

Templates. The next step is to create templates based on the topologies, this is done by the cloud expert. A template is a digital reflection of a topology including the attributes and some additional information such as node names and template

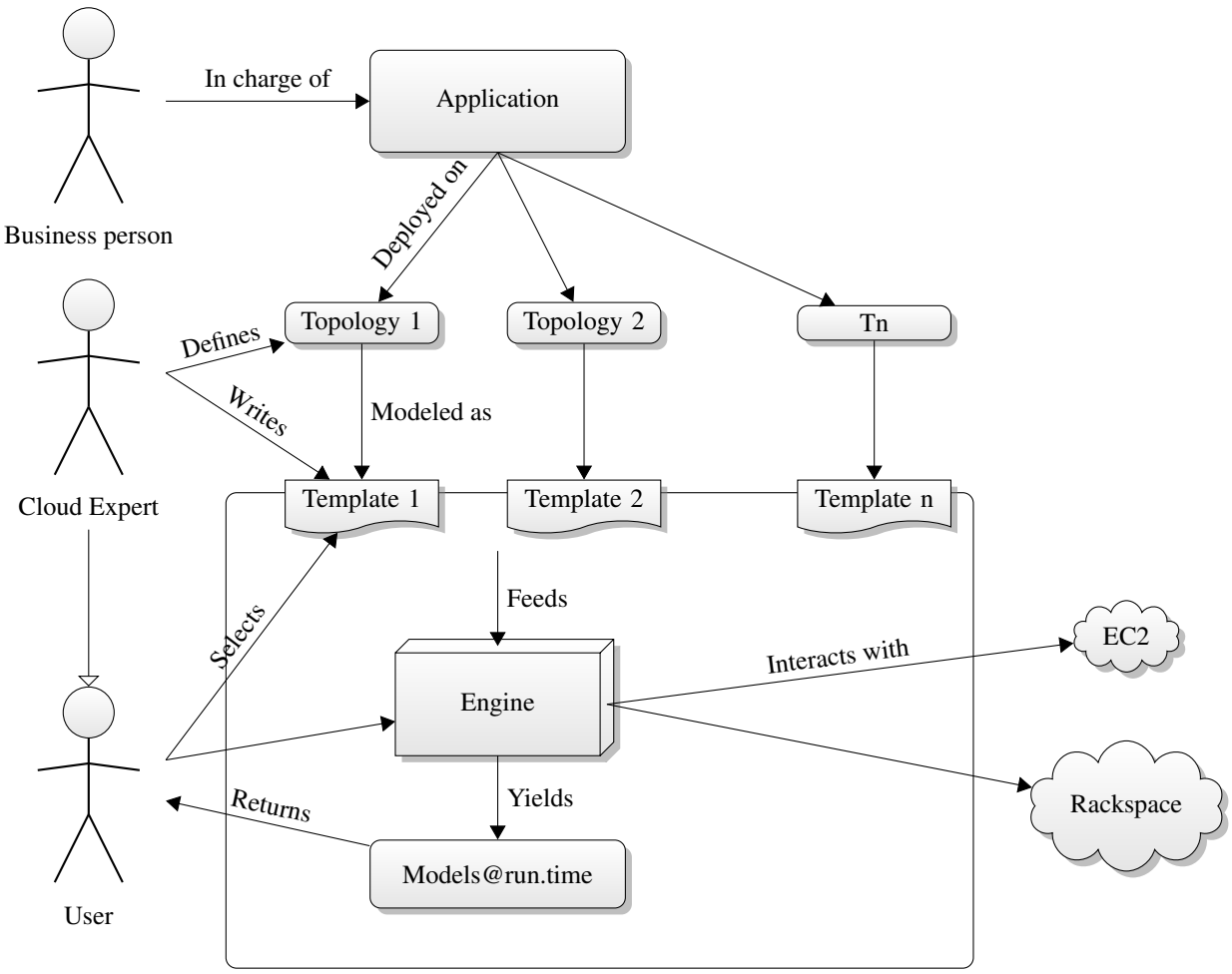

Figure 1.1: “Big picture”, overview of CloudML procedure flow.

labeling. It is also possible to define more than one topology within a single template.

Engine. When the cloud expert have designed and created the necessary templates the next actor, user, will continue the procedure. The user selects the template and feeds them into the engine. The engine is the core of the implementation, handling several steps and executing most of the CloudML logic. The engine operates according to five different steps:

1. Convert the template files into a native format for later use.
2. Convert pure nodes into instances ready for provisioning.
3. Connect to all the desired providers.
4. Propagate the instances.
5. Produce models@run.time of the instances being propagated.

Providers. The engine interacts with the providers, in FIG. **?? ac:EC2!** (**ac:EC2!**) and Rackspace are selected as examples, but any cloud provider that will be supported by CloudML can be utilized. As discussed in CHAP. **??** and CHAP. **??** different providers implement different solutions for communication *e.g.*, for provisioning nodes, managing nodes and services or terminating instances. For the engine to interact with a set of different providers, a tool, library or framework is needed. This additional software can connect to the different providers through a common interface.

Models@run.time. The last part of this implementation of CloudML (see CHAP. **??**) is to reflect provisioned instances with models@run.time. These models are returned to the user when provisioning starts, and when attributes and statuses about instances are updated the user is notified about these updates through the models. In the implementation the models can extend from or aggregate *instances*.

Part III

Conclusion