Chapter 1

Cloud computing and Model-Driven Artchitecture

Text here...

1.1 Cloud computing

Cloud computing is gaining popularity and more companies are starting to explore the possibilities as well as the limitation to the cloud.

Some of the most essential characteristics of cloud computing [5] are:

- *On-demand self-service*: Consumers can do provisioning without any human interaction. On-demand means dynamic scalability and elasticity of resource allocation, self-service means that users does not need to manually do these allocations themselves.
- *Broad network access*: Capabilities available over standard network mechanisms. Supporting familiar protocols such as HTTP/HTTPS and SSH.
- *Resource pooling*: Physical and virtual resources are dynamically assigned and reassigned according to consumer demand. This means users do not need to be troubled with scalability as this is handled automatically.

Provider	**Service**	**Service Model**
AWS	Elastic Compute Cloud	IaaS
AWS	Elastic Beanstalk	PaaS
Google	Google App Engine	PaaS
CA	AppLogic	IaaS
Microsoft	Azure	PaaS and IaaS
Heroku	Different services	PaaS
Nodejitsu	Node.js	PaaS
Rackspace	CloudServers	IaaS

Table 1.1: Common providers available services

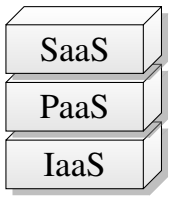

Figure 1.1: Cloud architecture service models

- *Rapid elasticity*: Automatic capability scaling. Already allocated resources can expand to meet new demands.
- *Measured service*: Monitoring and control of resource usages. Can be used for statistics for users but also for cloud services to do on-demand self-service, resource pooling and rapid elasticity.

There are three main architectural service models in cloud computing [5] namely *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS) and *Software-as-a-Service* (SaaS). IaaS is on the lowest vertical integration level closest to physical hardware and SaaS on the highest level as runnable applications. Stanoevska-Slabeva [8] emphasizes that *"infrastructure had been available as a service for quite some time"* and this *"has been referred to as utility computing"*, such as Sun Grid Compute Utility.

IaaS. The main providers are Google, Amazon with *Amazon Web Service* (AWS) [1] and Microsoft. A non-exhaustive list of common providers are visualized in TABLE. 1.1. The *National Institute of Standards and Technology* (NIST) is one of the leaders in cloud computing standardization. The NIST Definition of Cloud Computing [5] define IaaS as

> "The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.
>
> NIST, 2011

These are capabilities found in cloud provider services, such as AWS *Elastic Compute Cloud* (EC2) and Rackspace CloudServers. NIST continue to state that

> "The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (*e.g.*, host firewalls).
>
> NIST, 2011

PaaS. The PaaS model is defined as an capability consumers use to deploy applications onto cloud infrastructure that provide fully partially support. For this kind of deployment consumers do not have to manage or control underlying infrastructure capabilities, and in some cases not even configuration. Examples of PaaS providers are Google with *Google App Engine* (GAE) and the company Heroku with their service with the same name. Multiple PaaS providers utilize EC2 as underlying infrastructure, examples of such providers are Heroku Nodester and Nodejitsu, this is a tendency with increasing popularity.

SaaS. The core purpose is to provide whole web applications as services, in many cases end products. Google products such as gmail, Google Apps and Google Calendar are examples of SaaS applications.

There are four different deployment models according to The NIST Definition of Cloud Computing [5]:

- *Public cloud*: Infrastructure is open to the public. Cloud providers own the hardware and rent out IaaS and PaaS solutions to users. Examples of such providers are Amazon with AWS and Google with GAE.
- *Private cloud*: Similar to classical infrastructures where hardware and operation is owned and controlled by organizations themselves. This deployment model has arisen because of security issues regarding storage of data in public clouds. With *private cloud* organization can provide data security in forms such as geographical location and existing domain specific firewalls, and help complying requirements set by the government or other offices.
- *Community cloud*: Similar as *private clouds* but run as a coalition between several organizations. When several organizations share the same aspects of a private cloud (such as security requirements, policies, and compliance considerations), and therefore share infrastructure.
- *Hybrid cloud*: Combining private cloud or community cloud with public cloud. One benefit is to distinguish data from logic for purposes such as security issues, by storing sensitive information in a private cloud while computing with public cloud.

Beside these models defined by NIST there is another arising model known as *virtual private cloud*, which is similar to *public cloud* but with some security implications such as sandboxed network.

1.2 Model-Driven Architecture approach

By combining the world of cloud computing with the one of modeling it is possible to achieve benefits such as improved communication when designing a system and better understanding of the system itself. This statement is emphasized by Booch *et al.* in one of his studies:

> “Modeling is a central part of all the activities that lead up to the deployment of good software. We build models to communicate the desired structure and behavior of our system. We build models to visualize and control the system’s architecture. We build models to better understand the system we are building, often exposing opportunities for simplification and reuse. We build models to manage risk.”
>
> BOOCH, 2005

When it comes to cloud computing these definitions are even more important because of financial aspects since provisioned nodes instantly draw credit. The definition of “modeling” can be assessed from the previous epigraph, but it is also important to choose correct models for the task. Stanoevska-Slabeva emphasizes in one of her studies that grid computing “*is the starting point and basis for Cloud Computing.*” [8]. As grid computing bear similarities towards cloud computing in terms of vitalization and utility computing it is possible to use the same UML diagrams for IaaS as previously used in grid computing. The importance of this reusability of models is based on the origination of grid computing, *eScience*, and the popularity of modeling in this research area. The importance of choosing correct models is emphasized by Booch [2]:

> *(i)*The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped. *(ii)*Every model may be expressed at different levels of precision. *(iii)*The best models are connected to reality. *(iv)*No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models.
>
> BOOCH, 2005

These definition precepts state that several models (precept *(iv)*) on different levels (precept *(ii)*) of precision should be used to model the same system. From this it is concludable that several models can be used to describe one or several cloud computing perspectives. Nor are there any restraints to only use UML diagrams or even models at all. As an example AWS CloudFormation implements a lexical model of their *cloud services*, while CA AppLogic has a visual and more UML component-based diagram of their capabilities.

Model-Driven Architexture. When working with *Model-Driven Architecture* (MDA) it is common to first create a *Computation Independent Model* (CIM), then a *Platform-Independent Model* (PIM) and lastly a *Platform-Specific Model* (PSM). There are other models and steps in between these, but they render the essentials. There are five different life cycles as explained by Singh [7]:

1. Create a CIM capturing requirements.
2. Develop a PIM.
3. Convert the PIM into PSM.
4. Generate code form PSM.
5. Deploy.

Chapter 2

State of the Art in Provisioning

Evaluation of existing solutions
What have others done for multicloud provisioning
Even more examples in mOSAIC articles

- Identify *properties* (problems in reality)
- Find more sources
- Include more scientific literature!

2.1 Model driven

Amazon AWS CloudFormation http://aws.amazon.com

This is a service provided by Amazon from their popular Amazon Web Services. It give users the ability to create template files in form of JSON, which they can load into AWS to create stacks of resources. This makes it easier for users to duplicate a setup many times, and as the templates support parameters this process can be as dynamic as the user design it to be. This is a model in form or lexical syntax, both the template itself and the resources that can be used. For a company that is fresh in the world of cloud computing this service could be considered too advance. This is mainly meant for users that want to replicate a certain stack, with the ability to provide custom parameters. Once a stack is deployed it is only maintainable through the AWS Console, and not through template files. The format that Amazon uses for the templates is a very good format, the syntax is in form of JSON which is very readable and easy to use, but the structure and semantics of the template itself is not used by any other providers or cloud management tooling, so it can not be considered a multicloud solution. Even though JSON is a readable format, does not make it viable as a presentation medium on a business level.

CA Applogic http://www.3tera.com/AppLogic/

Applogic from CA is a proprietary model based web application for management of private clouds. This interface let users configure their deployments through

a diagram with familiarities to component diagrams with interfaces and assembly connectors. This is one of the solutions that use and benefit from a model based approach. They let users configure a selection of third party applications, such as Apache and MySQL, as well as network security, instances and monitoring. What CA has created is both an easy way into the cloud and it utilizes the advantages of model realizations. Their solution will also prove beneficial when conducting business level consulting. They also support a version of ADL (Architecture Deployement Language), a good step on its way to standardization. But this solution is only made for private clouds running their own controller, this can prove troublesome for migration, both in to and out of the infrastructure.

2.2 APIs

libcloud and jclouds http://libcloud.apache.org/ http://www.jclouds.org/

Libcloud is a API that aims to support the largest cloud providers through a common API. The classes are based around "Drivers" that extends from a common ontology, then provider-specific attributes and logic is added to the implementation. jclouds is very similar to libcloud but the API code base is written in Java and Clojure. This library also have "drivers" for different providers, but they also support some PaaS solutions such as Google App Engine. APIs can be considered modelling approaches based on the fact they have a topology and hierarchical structure, but it is not a distinct modelling. A modelling language could overlay the code and help providing a clear overview, but the language directly would not provide a good overview of deployment. And links between resources can be hard to see, as the API lacks correlation between resources and method calls. Libcloud have solved the multicloud problem in a very detailed manner, but the complexity is therefore even larger. The API is also Python-only and could therefor be considered to have high tool-chain dependency.

OPA http://opalang.org/

OPA is a cloud language aimed at easing development of modern web applications. It is a new language, with its own syntax, which is aimed directly at the web. The language will build into executable files that will handle load balancing and scalability, this is to to make this a part of the language and compilation. OPA is a new language, so it might be difficult to migrate legacy systems into this lanugage. There are no deployment configurations, as this is built into the language. The compiler will generate an executable that coWeb-based vs native application

The public cloud is located on the world wide web, and most of the managing, monitoring, payment and other administrative tasks can be done through web interfaces or APIs. Web applications are becoming more popular by the day, with HTML5, EcmaScript 5 and CSS3. The user experience in web applications today can in many cases match native applications, with additional benefits such as availability and ease of use. A web-based interface would prove beneficial for quickly displaying the simple core functionality of the language. In this era of cloud computing and cloud technologies a user should not need to abandon his or hers browser to explore the functionality of CloudML. Cloud providers are

most likely to give customers access to customize their cloud services through web-based interfaces, and if customers are to take advantage of CloudML, the language should be graphically integrated into existing tool chains. Providers would probably find it pleasing if a example GUI wauld be run on most cloud providers instances, and so it can also benefit from some cloud based load balancers, even though this is part of the language. The conclusion about OPA is that it is not a language meant for configuration, and could not easily benefit from a model based approach, and it does not intentionally solve multicloud.

Whirr http://whirr.apache.org/

This is a binary and code-based application for creating and starting short-lived clusters for hadoop instances. It support multiple cloud providers. It has a layout for configuration but it is mainly property-based, and aimed at making clusters.

Deltacloud http://incubator.apache.org/deltacloud/

Deltacloud has a similar procedure as jclouds and libcloud, but with a REST API. So they also work on the term "driver", but instead of having a library to a programming language the users are presented with an API they can call, on Deltacloud servers. This means users can write in any language they may choose. As well as having similar problems as other APIs this approach means that every call has to go through their servers, similar to a proxy. This can work with the benefits that many middleware software have, such as cahing, queues, redundancy and transformations, but it also has the disadvantages such as single point of failure and version inconsistencies.

2.3 Deployments

Amazon Beanstalk

simplifying-solution-deployment-on-a-cloud-through-composite-appliances

architecture-for-virtual-solution-composition-and-deployment

Chapter 3

Envision, concepts and principles

The core envision is to tackle challenges from CHAP. **??** by applying a model-driven approach supported by modern technologies. Main objective is to create a common model for nodes as a platform-independent model [6] to justify *multicloud* differences and at the same time base this on a human readable lexical format to resolve *reproducibility* and make it *shareable*.

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 envisioned to parse and execute template files representing topologies of instances in the cloud. Targeted users are application developers without cloud specific knowledge. The same files should be usable on other providers, and alternating the next deployment stack should be effortless. Instance types are selected based on properties within the template, and additional resources are applied when necessary and available. While the tool performs provisioning metadata of nodes is available. In the event of a template being inconsistent with possibilities provided by a specific provider this error will be informed to the user and provision will halt.

There are many cloud providers on the global market today. These providers support many layers of cloud, such as PaaS (Platform as a Service) and IaaS (Infrastructure as a Service). This vast amount of providers and new technologies and services can be overwhelming for many companies and small and medium businesses. There are no practical introductions to possibilities and limitations to cloud computing, or the differences between different providers and services. Each provider has some kind of management console, usually in form of a web interface and API. But model driven approaches are inadequate in many of these environments. UML diagrams such as deployment diagram and component diagram are used in legacy systems to describe system architectures, but this advantage has yet to hit the mainstream of cloud computing management. It is also difficult to have co-operational interaction on a business level without using the advantage of graphical models. The knowledge needed to handle one provider might differ to another, so a multicloud approach might be very resource-heavy on competence in companies. The types of deployment resources are different between the providers, even how to gain access to and handle running instances might be very different. Some larger cloud management application developers are not even providers themselves, but offer tooling for private cloud solutions. Some of these providers have implemented different types of web based applications that

let end users manage their cloud instances. The main problem with this is that there are no standards defining a cloud instance or links between instances and other services a provider offer. If a provider does not offer any management interface and want to implement this as a new feature for customers, a standard format to set the foundation would help them achieve a better product for their end users. These are some of the problems with cloud hosting today, and that CloudML will be designed to solve.

Chapter 4

Analysis and design - CloudML

- Copy chap 3 from CloudMDE
- Weaknesses

Just added everything here:

The meta model described in FIG. 4.1 and introduce CloudML by using a scenario where "Alice" is provisioning the *BankManager* to AWS *Elastic Compute Cloud* (EC2) using the topology shown in FIG. **??**. It is compulsory that she possesses an AWS account in advance of the scenario. She will retrieve security credentials for account and associate them with `Password` in FIG. 4.1. `Credential` is used to authenticate the user to supported providers through `Connector`. The next step for Alice is to model the appropriate `Template` consisting of three `Nodes`. The characteristics Alice choose for `Node Properties` are fitted for the chosen topology with more computational power for front-end `Nodes` by increasing amount of `Cores`, and increased `Disk` for back-end `Node`. All `Properties` are optional and thus Alice does not have to define them all. With this model Alice can initialize provisioning by calling `build` on `CloudMLEngine`, and this will start the asynchronous job of configuring and creating `Nodes`. When connecting front-end instances of *BankManager* to back-end instances Alice must be aware of the back-ends `PrivateIP` address, which she will retrieve from CloudML during provisioning according to *models@run.time* (M@RT) approach. `RuntimeInstance` is specifically designed to complement `Node` with `RuntimeProperties`, as `Properties` from `Node` still contain valid data. When all `Nodes` are provisioned successfully and sufficient metadata are gathered Alice can start the deployment, CloudML has then completed its scoped task of provisioning. Alice could later decide to use another provider, either as replacement or complement to her current setup, because of availability, financial benefits or support. To do this she must change the provider name in `Account` and call `build` on `CloudMLEngine` again, this will result in an identical topological setup on a supported provider.

Implementation. CloudML is implemented as a proof of concept framework [3] (from here known as *cloudml-engine*). Because of Javas popularity cloudml-engine was written in a JVM based language with Maven as build tool. Cloudml-engine

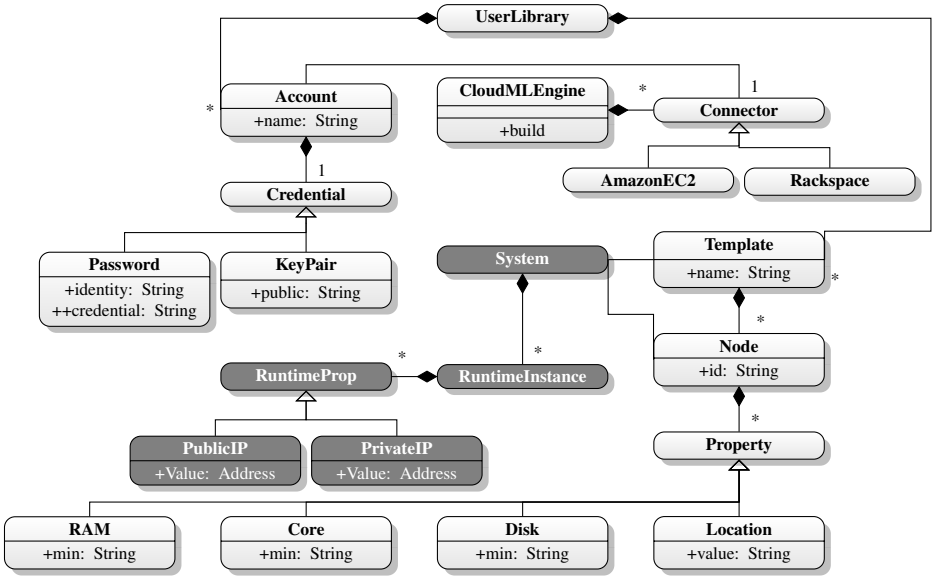

Figure 4.1: Architecture of CloudML

use jclouds.org library to connect with cloud providers, giving it support for 24 providers out of the box to minimize *complexity* as well as stability and *robustness*.

Provisioning nodes is by its nature an asynchronous action that can take minutes to execute, therefore CloudML relied on the actors model [4] using Scala actors. With this asynchronous solution CloudML got concurrent communication with nodes under provisioning. The model is extended by adding a callback-based pattern allowing each node to provide information on property and status changes. Developers exploring the implementation can then choose to "listen" for updating events from each node, and do other jobs / idle while the nodes are provisioned with the actors model. The terms are divided for a node before and under provisioning, the essential is to introduce *M@RT* to achieve a logical separation. When a node is being propagated it changes type to `RuntimeInstance`, which can have a different *state* such as *Configuring*, *Building*, *Starting* and *Started*. When a `RuntimeInstance` reaches *Starting* state the provider has guaranteed its existence, including the most necessary metadata, when all nodes reaches this state the task of provisioning is concluded.

Full deployment is planed for next version of CloudML.

Bibliography

[1] Amazon. Amazon web services, 2012.

[2] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[3] Eirik Brandtzæg. cloudml-engine, 2012.

[4] Philipp Haller and Martin Odersky. Actors that unify threads and events. In *Proceedings of the 9th international conference on Coordination models and languages*, COORDINATION'07, pages 171–190, Berlin, Heidelberg, 2007. Springer-Verlag.

[5] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, 145(6):7, 2011.

[6] Viet Cuong Nguyen and X. Qafmolla. Agile Development of Platform Independent Model in Model Driven Architecture. In *Information and Computing (ICIC), 2010 Third International Conference on*, volume 2, pages 344 –347, june 2010.

[7] Y. Singh and M. Sood. Model Driven Architecture: A Perspective. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 1644 – 1652, march 2009.

[8] Katarina Stanoevska-Slabeva and Thomas Wozniak. Cloud Basics - An Introduction to Cloud Computing. In Katarina Stanoevska-Slabeva, Thomas Wozniak, and Santi Ristol, editors, *Grid and Cloud Computing*, pages 47–61. Springer Berlin Heidelberg.