Use of Modeling Language to deploy applications in clouds.

Gurpreet K. Juneja

India

***Abstract*— Resource Sharing in a pure plug and play model that dramatically simplifies infrastructure planning is the promise of cloud computing. The two key advantages of this model are ease-of-use and cost-effectiveness. Cloud Computing also offers vast amount of resources available for end users. The opportunity to choose between several cloud providers is alluded by complexity o cloud solution heterogeneity. Challenges with cloud deployment and resource provisioning are identified in this paper. To tackle these challenges a model based language named cloud ML is proposed. This language is supported by an engine able to provision nodes in the cloud.**

***Keywords*— MDA, CIM, PIM, PSM, Template, Instance, Node.**

I. Introduction To Cloud Computing

Cloud Computing is a computing paradigm where a large pool of systems are connected in public or private networks to provide dynamically scalable infrastructure for application, data and file storage. With the advent of this technology the cost of computation, application hosting, content storage and delivery is reduced significantly. This is shown in fig. 1.

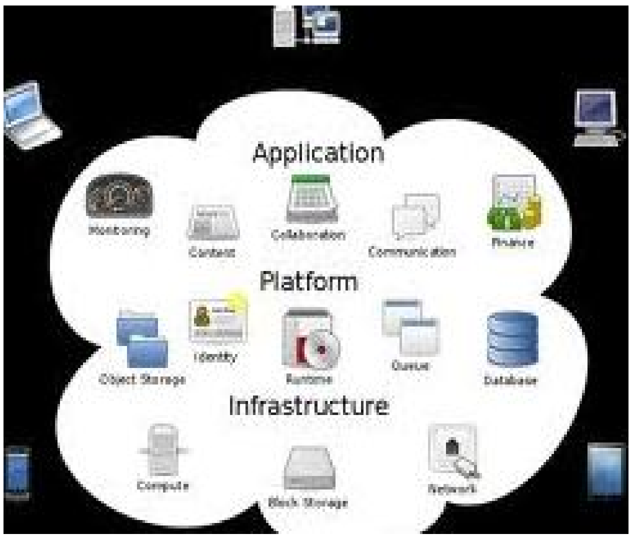

Fig. 1 Cloud Computing

A. Service Models

Cloud Providers offers services that can be grouped into three categories.

1) Software as a Service (SaaS): In this model, a complete application is offered to the customer, as a service on demand. A single instance of the service runs on the cloud & multiple end users are serviced. On the customers‟ side, there is no need for upfront investment in servers or software licenses, while for the provider, the costs are lowered, since only a single application needs to be hosted & maintained. Today SaaS is offered by companies such as Google, Microsoft, Zoho, etc.

2) Platform as a Service (PaaS): Here, a layer of software, or development environment is encapsulated & offered as a service, upon which other higher levels of service can be built. The customer has the freedom to build his own applications. Eg. Google app engine.

3) Infrastructure as a Service (IaaS): IaaS provides basic storage and computing capabilities as standardized services over the network. Servers, storage systems, networking equipment, data centre space etc. are pooled and made available to handle workloads. The customer would typically deploy his own software on the infrastructure. Some common examples are Amazon, GoGrid, 3 Tera, etc.

B. Deployment Models: Deployment models define where and how applications are deployed in a cloud environment, such as publicly with a global provider or private in local data centres. There are four main deployment models.

1) Public cloud: A public cloud can be accessed by any subscriber with an internet connection and access to the cloud space.

2) Private cloud: A private cloud is established for a specific group or organization and limits access to just that group.

3) Community cloud: A community cloud is shared among two or more organizations that have similar cloud requirements.

4) Hybrid cloud: A hybrid cloud is essentially a combination of at least two clouds, where the clouds included are a mixture of public, private, or community.

II. Model Driven Engineering

By combining the domain 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. Model-Driven Architecture (MDA) is a way of designing software with modeling in mind provided by the Object Management Group (OMG). This is shown in the fig below.

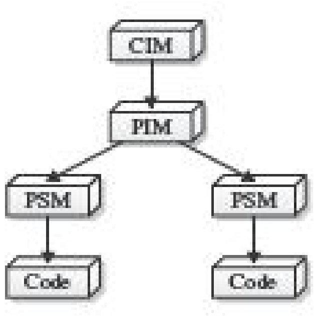

Fig. 2 Model driven architecture

When working with 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 five different steps which are discussed below.

A. *Create a CIM*

This is done to capture requirements and describe the domain. To do this the MDA developer must familiarize with the business organization and the requirements of this domain. This should be done without any specific technology. The physical appearance of CIM models can be compared to use case diagrams in UML, where developers can model actors and actions (use cases) based on a specific domain.

B. *Develop a PIM*

The next step aims at using descriptions and requirements from the CIM with specific technologies. The OMG standard for MDA use UML models, while other tools or practices might select different technologies. Example of such Platform Independent Models can be class diagrams in UML used to describe a domain on a technical level.

C. *Convert PIM into PSM*

The next step is to convert the models into something more concrete and specific to a platform.

D. *Generate code from PSM*

A PSM should be specific enough that code can be generated from the models. For instance can class diagrams be generated into entities, and additional code for managing the entities can be added as well. Some diagrams such as Business Process Model and Notation (BPMN) can generate Business Process Execution Language (BPEL) which again can generate executable logic.

E. *Deploy*

The final step is based on deploying the PSM, which concludes the five steps from loosely descriptions of a domain to a running product.

III. Challenges Posed By Cloud Computing

For all the benefits that cloud computing promises, it also poses a number of challenges for providers and consumers. The main hurdles to be negotiated lie in sufficient utilisation of the IT capacities, contractual complexities, regulations on data access, the concentration of data and the fact that the user is tied to one cloud provider. The major challenges are discussed below.

A. *Aiming for optimum capacity utilisation*

To achieve optimum utilisation of their IT infrastructure, they therefore usually try to acquire an ideally complementary customer portfolio of users diversified across individual sectors and time zones.

B. *Contract terms often opaque*

The contractual relationship between cloud computing vendors and users is often not set out in full, particularly since the contracting parties frequently fail to negotiate a sufficiently comprehensive agreement. In many instances users have absolutely no idea who is actually delivering the service at the end of a long value chain of sub-contractors. In such cases a legally enforceable contractual relationship can normally be deemed to exist only indirectly.

C. *Agreement on the service quality required*

In a sustainable partnership the parties involved should agree on the minimum level of service quality (with regard to system availability and speed) that is to be observed. In practice, however, the issue of quality is frequently left open in the service level agreement.

D. *Data protection and data security*

In addition to the quality of service, the varied aspects of data protection and data security are also extremely important for the cloud consumer when drawing up a full and complete agreement.

IV. Vision, Concepts And Principles of Cloud ML

The concept and principle of Cloud ML 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 domain of cloud ML contains components necessary to implement in order to fulfil 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. Various parts and components of cloud ML is shown diagrammatically and is discussed below.

A. *Actors*

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 utilizes CloudML to do provisioning. This physical person may or may not have the role of cloud expert, hence the cloud expert extends from the user actor.

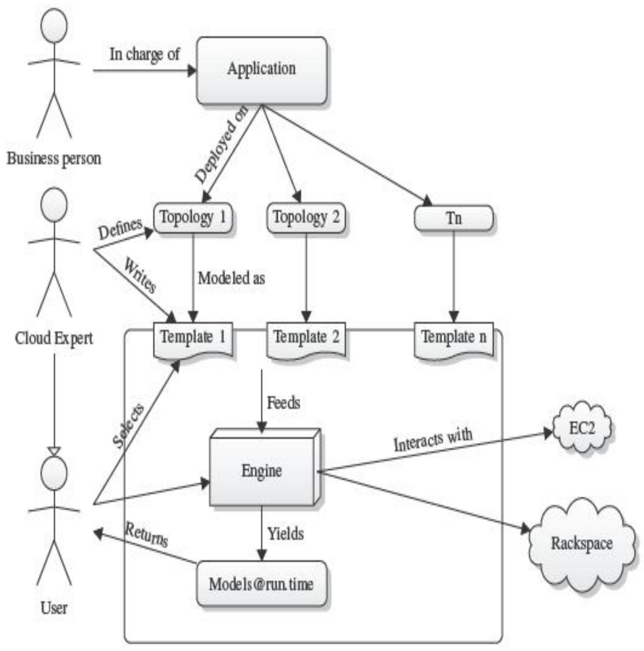

Fig. 3 Domain of cloud ML

B. *Application and Topologies*

The business person is in charge of the application, he/she has a need for an application that can fulfil 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., CPU power and RAM sizes. He/she might create several topologies to fulfil the application demands.

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

D. *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 first of all converts the template files into a native format for later use. It then, convert pure nodes into instances ready for provisioning. Then, it connects to all the desired providers. After that it propagates the instances and produces models @run time of the instances being propagated.

E. *Providers*

The engine interacts with the providers, EC2 and Rackspace are selected as examples, but any cloud provider that will be supported by CloudML can be utilized. 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.

F. *Models@ run time*

The last part of this implementation of CloudML 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.

V. Analysis and Design- Cloud ML

For analysis and design considerations of cloud ML, let us take the example of Alice by considering fig. 3. Alice here performs the role of cloud expert and user. She will define the topologies, create the templates and use the engine to provision her models.

A. *Single node topology*

Since this single node handles both computation and storage, Alice decides to increase capabilities of both processing (number of Cores) and Disk size on the Node. Both of these attributes are incremented because the instance hosts the main application as well as the database. The approach of using one single node is good in terms of simplicity. This is shown in the fig below.

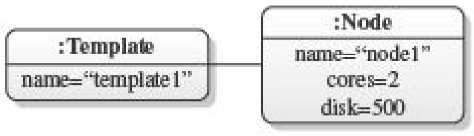

Fig. 4(a) Template and Node

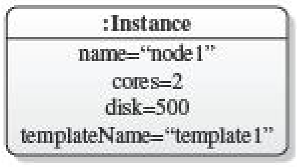

Fig. 4(b) Instance

B. *Building Templates*

In the end Alice inserts all data about topologies into a Template. The template includes physical descriptions of the Node, and a list of the type Property for the node. The Node has a name used to reference the node under provisioning. The properties the node can have are configurations of attributes on a set of given capabilities. These configurations are what define what type of tasks a node is suitable for. In Alice's case the node has increased two important attributes to support both higher computation demand and storage capabilities, i.e., 2 cores and 500 Gigabyte (GB) 1 in hard drive size. By not altering any other attributes on the respective nodes, they will be set to minimal values. This is a positive expectation, since the nodes will handle specific tasks, which does not demand enhancing of other attributes.

C. *Provisioning*

With these models Alice initializes provisioning by calling build on Cloud ML Engine, providing Credential and Template. This starts the asynchronous job of configuring and creating Instances based on nodes. Provisioning process is shown in the form of following sequence diagram.

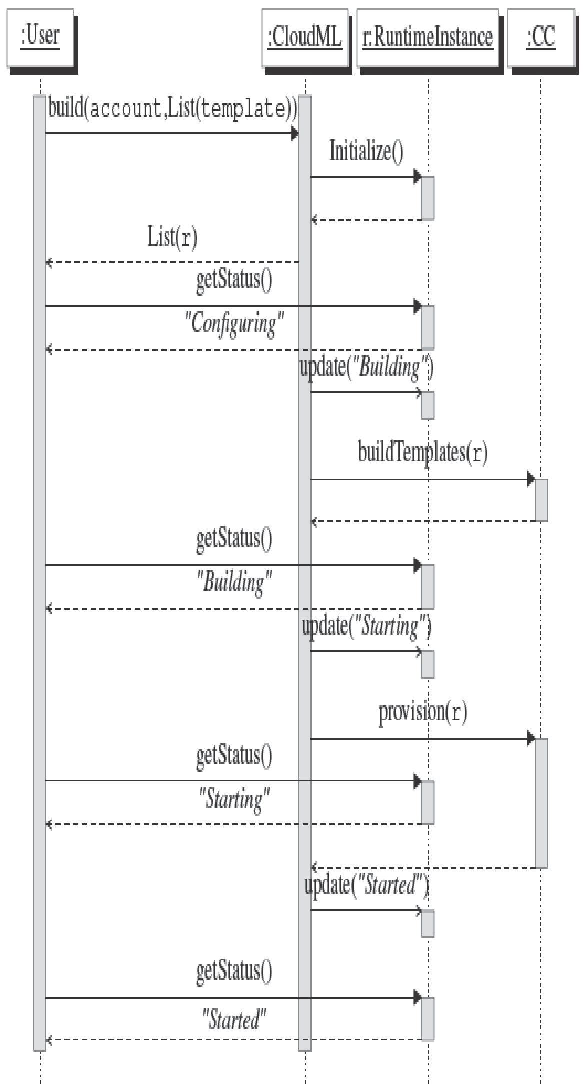

Fig. 5 Cloud ML provisioning process (Sequence diagram)

Here, Instance only refers to template by a String, templateName. This is semantically correct because the template is a transparent entity in the context of provisioning, and is only used as a reference. Instance is also an internal element in CloudML, and might not have to be indirectly or directly exposed to end users. RuntimeInstance is specifically designed to complement Node with RuntimeProperties, as Properties from Node still contain valid data. When CloudMLEngine start provisioning, a RuntimeInstance is created immediately, and returned to Alice. The method call to build is described in fig. In this figure RuntimeInstance is returned directly to Alice, because these are asynchronous elements within CloudML. The actor CC within this figure is an abbreviation of CloudConnector. This is emphasized in fig. through getStatus method calls.

D. *Three nodes topology*

For scalability and modularity the single-node approach is restraining, i.e., it does not scale very well, and does not benefit from cloud advantages. If the application consumes too much CPU power, this slows the application totality down and decreases usability. There is no strong link between CloudML and the application, but to maintain scalability some measures must be manually developed using three nodes topology. This is shown in the following figure.

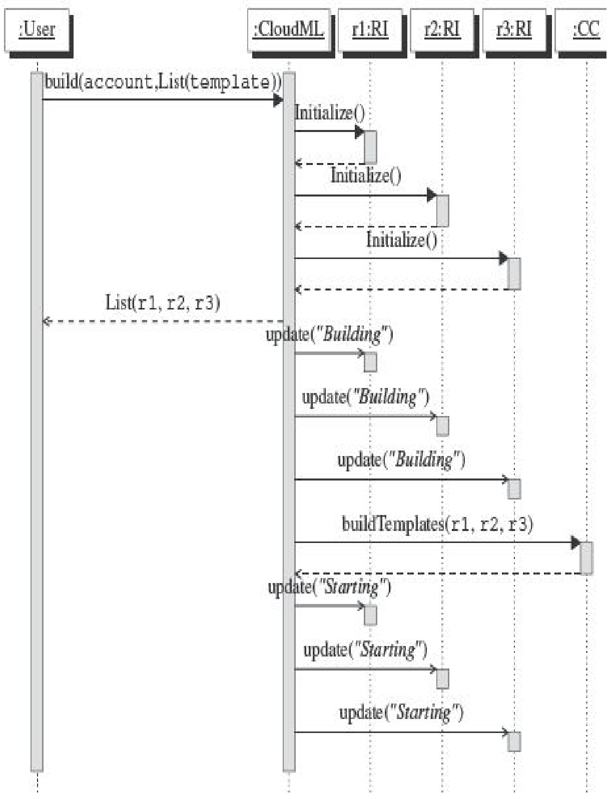

Fig. 6 Three nodes provisioning process

E. *New template*

Alice changes her topology by editing her existing template to contain three nodes instead of one. She also changes the node attributes to suite their new needs better, i.e., increasing amount of Cores on front-end, and increased Disk for back-end Node. Three nodes topology is shown in the fig below.

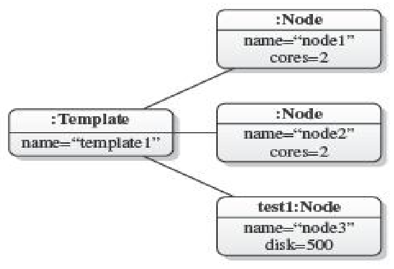

Fig. 7(a) Template with nodes

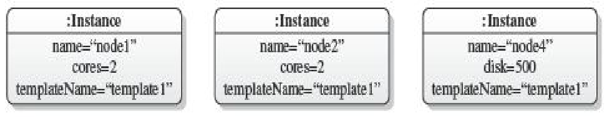

Fig. 7(b) Instance

So, the benefit of a three node topology where the application is distributed over several nodes is the scalability and modularity, which were lacking in the single-node topology.

VI. CONCLUSION

First the background part introducing the domain of cloud computing and model-driven engineering which combines the domain of Cloud Computing with the one of modeling to achieve benefits such as improved communication when designing a system and better understanding of the system itself. Then, various challenges with cloud deployment and resource provisioning are identified. To tackle these challenges a model based language named cloud ML is proposed. After that, the core idea of CloudML is introduced. Finally, analysis and design of Cloud ML should be built up. All through is done with the help of a scenario where Alice performs provisioning with the help of single node topology. But, for scalability and modularity, three nodes topology is used where the application is distributed over several nodes.

.

REFERENCES

[1] S. Mosser, B. Eirik and P. Mohagheghi, *Cloud-Computing: from Revolution toEvolution*, BElgian-NEtherlands software eVOLution seminar (BENEVOL'11), pages 1–2, Brussels, Belgium, December 2011.

[2] E. Brandtzæg, P. Mohagheghi and S. Mosser, *Towards a Domain-Specific Language to Deploy Applications in the Clouds*, Third International Conference on Cloud Computing, (CLOUD'12), July 2012.

[3] A. Huth and J. Cebula, *The Basics of Cloud Computing*, United States-Computer Emergency Readiness Team (US-CERT), 2011.

[4] T. Harris, *Cloud Computing: An Overview*, White Papers DB.

[5] S. Heng, *Cloud Computing- clear skies ahead*, E-conomics Digital economy and structural change, March 1, 2012.

[6] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide*, The (2nd Edition) (Addison-Wesley Object Technology Series). Addison-Wesley Professional, 2005. ISBN 0321267974.

[7] E. Brandtzæg, *Bank manager*, 2012. URL https://github.com/eirikb/grails-bank-example.

[8] P. Haller and M. Odersky, *Actors that unify threads and events*, Proceedings of the 9th international conference on Coordination models and languages, COORDINATION'07, pages 171–190, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72793-4.

[9] S. Kent, *Model Driven Engineering*, Integrated Formal Methods, volume 2335 of Lecture Notes in Computer Science, pages 286–298. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-43703-1.

[10] Rackspace, *Rackspace cloud*, 2012. URL http://www.rackspace.com/cloud.

[11] Y. Singh and M. Sood, *Model Driven Architecture: A Perspective*. Advance Computing Conference, 2009. IACC 2009. IEEE International, pages 1644 –1652, march 2009. doi: 10.1109/IADCC.2009.4809264.

[12] J. Varia, *Architecting for the Cloud: Best Practices*, Compute, 1(January):1–23, 2010.