An Architecture for a Heterogeneous Private IaaS Management System

Rodrigo García-Carmona, Mattia Peirano, Juan C. Dueñas, Álvaro Navas
Departamento de Ingeniería de Sistemas Telemáticos
ETSI Telecomunicación, Universidad Politécnica de Madrid
Madrid, Spain
rodrigo@dit.upm.es, peirano.m@gmail.com, jcduenas@dit.upm.es, anavas@dit.upm.es

***Abstract*—Cloud computing and, more particularly, private IaaS, is seen as a mature technology with a myriad solutions to choose from. However, this disparity of solutions and products has instilled in potential adopters the fear of vendor and data lock-in. Several competing and incompatible interfaces and management styles have given even more voice to these fears. On top of this, cloud users might want to work with several solutions at the same time, an integration that is difficult to achieve in practice. In this paper, we propose a management architecture that tries to tackle these problems; it offers a common way of managing several cloud solutions, and an interface that can be tailored to the needs of the user. This management architecture is designed in a modular way, and using a generic information model. We have validated our approach through the implementation of the components needed for this architecture to support a sample private IaaS solution: OpenStack.**

Keywords-private IaaS; cloud management; management architecture; cloud interoperability; OpenStack.

I. INTRODUCTION

Cloud computing has, during recent years, gained traction both in the enterprise world and the academia. Among all possible cloud service models, one in particular, the private IaaS, has experienced an exceptional growth in the number of solutions available [1]. Several competing products, both open-sourced and proprietary, are contending for attaining relevance and are constantly trying to surpass each other. This fact creates a climate in which the user has the possibility of choosing among a huge array of possible solutions.

However, this ample offer of private IaaS cloud technologies also involves an important drawback: each one is managed using different abstractions (sometimes for the same concepts) and through different management interfaces. This is aggravated by the use of different technologies for these interfaces. This presents problems for a more widespread adoption of private IaaS cloud computing, since potential users fear of being locked-in with a particular solution that falls behind the others in terms of features or support. The infrastructure's owner should be able to change his previously chosen technology for private IaaS without having to modify the management interfaces, a fact that sometimes incur in expensive retraining and even more expensive errors during production deployments. Vendor and data lock-in are considered two of the bigger factors that hinder the development of cloud computing [2] [3].

Moreover, an enterprising private IaaS user could have the desire of deploying two or more different cloud offerings, leveraging the strong features of each for a solution better tailored to his or her specific needs. In this situation the user would benefit greatly from an integrated management interface that could wrap this mixture of products in a uniform whole. Another reason for deploying two private IaaS solutions at the same time is to compare their performance side to side or ease the migration from one to the other.

With this problem in mind, we propose a generic management system for private IaaS clouds, decoupled from any particular solution but able to work with all of them, and using a set of common abstractions that could be translated to the specifics of each targeted product. This management system should also be able to provide its interface through the use of different technologies (like a REST web service, a command line, or a web page), to better suit the user's needs.

To achieve this goal, we have defined a modular architecture, in which components for both different private IaaS technologies and interfaces can be developed and plugged as needed. In this paper, we present this architecture, validating it through the implementation of the components needed for the management of OpenStack clouds.

The next section of this paper features a brief view of existing private IaaS management solutions and related research. After it, in Section 3, we show the general architecture of the proposed management system. Section 4 covers the specifics related to the OpenStack implementation of the management system. Finally, the last section of this paper summarizes our achievements and what was learned in the experience, while also exposing some possible future lines of work.

II. PRIVATE IAAS MANAGEMENT SOLUTIONS

There are a multitude of management interfaces for cloud infrastructure and storage services. Every solution has at least one, and they can be found in multiple shapes: command-line tools, locally installed management applications with a GUI, web browser extensions, online tools, etc.

 ISBN: 978-1-61208-271-4

All private IaaS solutions offer their own management interfaces, tailored to its specific needs and features, and rarely able to interact with other solutions, or even other cloud deployments of the same solution. The only exceptions to this fact are not by design: it is just that some private IaaS solutions try to replicate the same capabilities and abstractions offered by more popular public offerings, like Amazon AWS. And, in doing so, they develop very similar or even identical interfaces. Among these the more extended are Eucalyptus, Nimbus, OpenNebula and OpenStack. Comparisons between the IaaS solutions usually include a comparison between their management interfaces [4]–[6]. All considered, these management systems are usually solutions particularized to work only with a specific cloud technology, and they are not compatible with others.

Third party solutions for managing private IaaS clouds also exist, some of them suited to just one cloud solution [7], while others support several technologies. KOALA (Karlsruhe Open Application (for) cLoud Administration) is a web based application able to manage and control AWS compatible cloud services [8]. It allows to work with a large variety of services of various public and private cloud providers in a seamless and transparent way [9]. KOALA innovative characteristic is that it does not require a local installation since itself could be deployed in the cloud. The user interface allows customers to start, stop and monitor their instances or volumes in various cloud infrastructure regions, and have access to the console output of virtual machines. KOALA supports S3, Google Storage and Walrus storage services.

Scalr is a cross platform, cloud management software that provides auto scaling disaster recovery and server management [10]. It is open source, available at Google Code but a hosted version is available as paid service. The manager is able to scale the virtual infrastructure according to the load. Scaling strategies could be based on CPU, RAM, disk, network or date. The latter can be useful in case of an increase in traffic is expected, like during scheduled public events. The code is distributed under Apache 2 license.

Puppet is an IT automation software that helps system administrators manage infrastructure throughout its lifecycle, easing the automation of the repetitive tasks [11]. This configuration management tool is written in Ruby and provides some specific modules for cloud management. The software is distributed for free with some utilization restrictions. The paid version offers a solution without limits.

Finally, there are open source initiatives like Libcloud [12], jcloud [13] or deltacloud [14], but they are more concerned with the management of public IaaS providers, even if they include support for some private IaaS solutions. They are centred in the management of virtual instances, and do not give much attention to the physical underlying infrastructure while doing so. Also, they are limited to a specific programming language or interface.

As can be seen, none of them offers a true solution-agnostic view. The academia has produced systems that offer a generic management interface that could potentially be adapted to any particular product. However, they present an important drawback: their interface is fixed, since their generic model and interface are built with a specific technology in mind, like REST [15] or SOAP [16] web services. Therefore, it is difficult to extend them to provide other kind of interfaces, like command-line tools or web pages.

In the end, the fact that the existing management interfaces are focused in exposing low-level infrastructure elements makes working with several solutions or migrating from one to another a complex affair, since there is no exact match between the features and abstractions being used by each one [17]. This is mainly because they are focused in the management of resources, not applications.

III. Management system architecture

The first step for developing a technology-independent management system is to have a generic information model that covers all the elements that need to be managed in a private IaaS. We have developed a model that bridges the gap between the applications deployed into a cloud and the actual resources allocated to them. This way the user of the management infrastructure can have a view of the big picture. This model has already been accepted for publication [18] and, therefore, will not be the topic of this paper. However, its elements related to the IaaS resources are summarized in Figure 1. They are mostly self-explanatory, but it is important to note the fact that all elements are *Resources* (and because of that are managed uniformly), and that there are relationships between the physical and virtual *Resources*, enabling traceability. The *VirtualMachine* element represents the VMs managed.

This model is able to cover every solution, but none of them are able to work with it without modification; a transformation from this model to the internal representation specific for each cloud technology is necessary. Similarly, a translation between the generic management actions and the operations enabled by each cloud technology must be provided. This two tasks are fulfilled by the management system, and these needs determine its architecture, which is shown in Figure 2. The management system is divided in three separate layers:

- The topmost layer, named *Control Layer*, is responsible of providing an interface to the outer world, and makes use of the set of services provided by the *Management Layer* to execute the management actions.
- Under it lies the *Management Layer*, which is the heart of the system. This layer is composed by a set of interfaces that define the capabilities of the system and one or more implementations of them, providing management over specific functional areas of a particular private IaaS cloud technology.

 ISBN: 978-1-61208-271-4

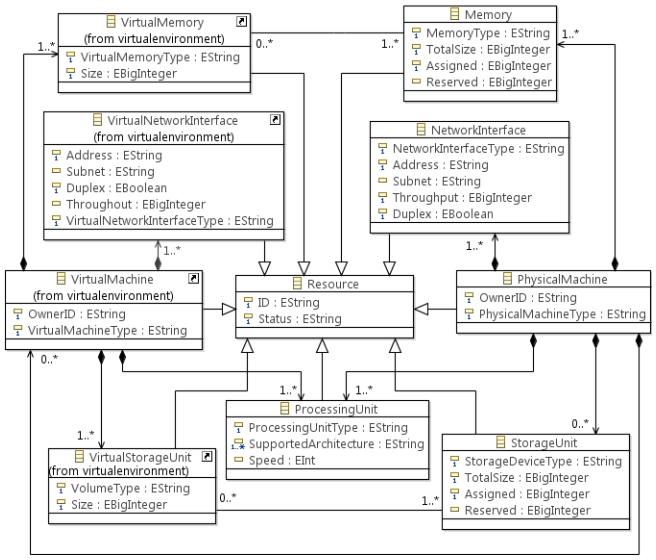

Figure 1. Generic Information Model

- The lowest layer is the *Client Layer*, which connects the system to the cloud solution itself.

The target of this division is to support a) several interfaces for the same management system, b) several cloud solutions, and c) several client technologies for the same cloud solution. We will analyse each of this layers in detail in the following subsections.

A. Control Layer

This layer is divided in 2 other: the *Control Interface* and the *High-Level Managers*.

The *Control Interface* is the outward interface that connects the system to the manager. This layer is designed to be interchangeable and support several interface implementations at the same time. The motivation for this schema is that different users could prefer different management interfaces. Moreover, the user does not need to be a human operator at all, it can be other system, and therefore there is a need for interfaces more suited to this task. Samples of these interfaces could be command-line tools or an administration web page for a human operator, and a web services interface for an autonomic system that keeps care of the private cloud infrastructure.

Under the *Control Interface* lies the *High Level Managers*, which intermediate between the aforementioned interface implementations and the *Management Layer*. The *High Level Managers* sublayer provides to the *Control Interface* two components: an *Infrastructure Manager* for controlling the cloud through a set of management actions defined in our information model (and therefore technology-agnostic), and an *Authentication Manager* in charge of monitoring and enforcing the security model for the private cloud. This component deserves to be separated from the *Infrastructure*

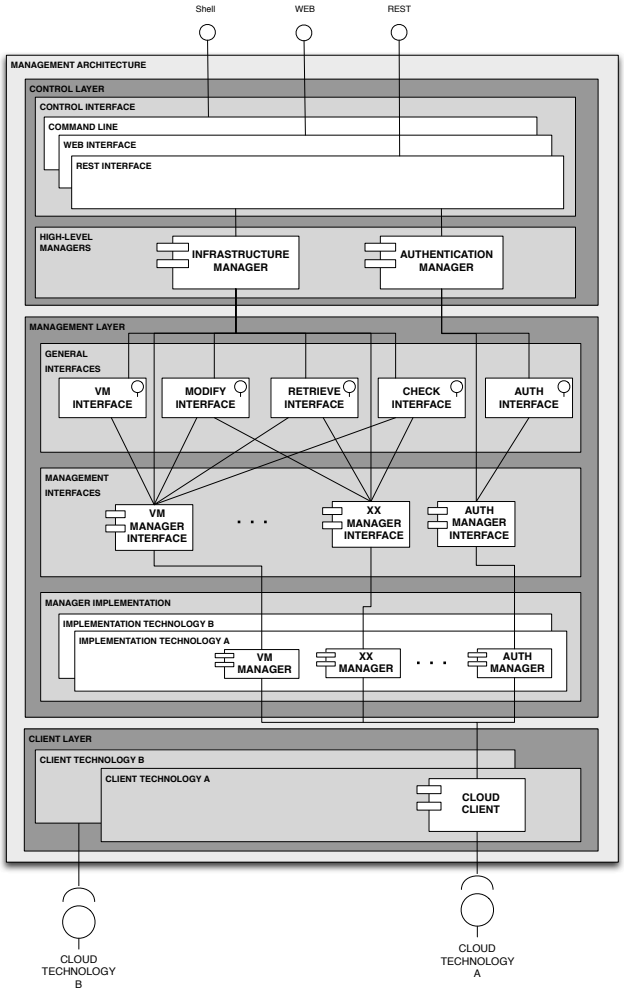

Figure 2. Management System Architecture

Manager because of the high importance of security in a cloud environment, where there could be multiple tenants, and the different implementations of the security system that each cloud solution features. It is necessary to have a common interface that abstracts from this differences and complexities.

B. Management Layer

This layer is again divided in 3 other: *General Interfaces*, *Management Interfaces* and *Manager Implementations*.

The *General Interfaces* sublayer offer three interfaces which provide management primitives that can be applied to every *Resource* as defined in our information model. These primitives are very simple, and the more complex management activities are built upon them:

- *Retrieve*, which encompasses the actions to obtain data from the cloud: *getList* and *getSpecific*.
- *Modify*, which includes the actions tailored to modify

the state of the cloud: *create* and *delete*. Modification is just a deletion followed by a creation.

- *Check*, used to check if a element already exists in the cloud. It includes just one action of the same name: *check*. This feature is needed to be kept up to date with the state of the environment.

The *General Interfaces* also include two other interfaces:

- *VM*, for controlling Virtual Machines, one of the most important elements of our model. It provides the actions *Suspend*, *Resume*, *Resize* and *Migrate*.
- *Authentication*, which does not perform actions over entities at all. Instead, it controls who can perform them and under what circumstances.

Under the *General Interfaces* lie the *Management Interfaces*. This sublayer particularises the primitives of the upper level to specific *Resources* of the infrastructure. We have defined the following interfaces, whose responsibility can be easily inferred from their names: *VM Manager*, *Image Manager*, *Virtual Appliance Manager*, *Compute Manager*, *Network Manager*, *Host Manager*, *Key Manager*, *Group Manager*, *Tenant Manager* and *Authentication Manager*. These interfaces extend one or more of the *General Interfaces* level interfaces.

These interfaces are in turn implemented in the remaining sublayer: *Manager Implementation*. Since this implementation has to be tailored for each cloud technology, it is here where the adaptation between our generic information model and the solution's specific one is performed. Therefore, we must implement the *General Interfaces* for each cloud technology we desire to support. If this is done correctly several solutions could be used at the same time, without each one being conscious of the others.

C. Client Layer

Finally, in the *Client Layer* is where the adaptation between the management system and the real infrastructure is realised. This is done through one or more modules, each designed to work with a particular access technology and cloud solution. These modules connect each with the appropriate *Manager Implementation* and interact with the private IaaS itself.

IV. OpenStack Management

To validate our proposal, we decided to create an implementation of the management architecture able to interface with one of the existing private IaaS solutions: OpenStack. We chose this product because its open sourced nature would help us in solving any problems that might arise. On top of that, it is a relatively mature solution that is seeing intense development at the moment.

To adapt a cloud solution to our proposal, we had to complete three tasks: 1) specify a translation between our generic information model and the solution's own, 2) develop a corresponding set of *Manager Implementations*, and 3) create at least one interface for the *Client Layer*.

TABLE I. Mapping Between Information Models

Generic Model	OpenStack Model
Virtual Instance	Virtual Machine (*Server*)
Virtual Appliance	Image
	Flavor
	Name
	Security Group
	Metadata
Virtual Memory	Flavor (*RAM*)
Virtual Storage	Flavor (*Disk*)
	Volume
Processing Unit	Flavor (*CPU*)
Virtual Network Iface.	Virtual Network
Owner	Tenant
Physical Machine	Host
-	User
Initial Configuration	Key Pair
-	Floating IPs

Table I shows the mapping between our generic information model and the OpenStack representation that we developed. Each column includes some elements that are not present in the other. For example the *Virtual Appliance* element is not defined in the OpenStack environment. However, a correspondence between several disparate elements of the OpenStack model to it can be made. Even if they are placed in different relative places inside their own model, most components of every private IaaS can be traced to elements of our information model. There are exceptions, though, like the *User* and *Floating IPs*. But in these cases the culprits are always relevant to specifics of each implementation, and can be managed inside the *Manager Implementation* sublayer without hampering the view of the environment.

After the mapping is defined the grunt work of the adaptation to OpenStack lies in the development of the *Manager Implementations* and *Client Layer*. Most of this work is just programming and is no relevant to this text, but one aspect of it took a special importance during the process: an OpenStack component named Quantum. Quantum provides advanced high level network management, enabling the definition of L2 and L3 network topologies and multiple networks across different VMs and tenants. Quantum was still in its early stages of development while we were validating our proposal and, in fact, there was no complete support for it in the OpenStack interface. Therefore, the development of our *ClientLayer* involved modifying the OpenStack code itself. Also, Quantum forced us to rethink some aspects of the network-related elements in our model, to support its more advanced capabilities. The *Client Layer* was designed to use the OpenStack REST interface.

To illustrate how the different managers interact among themselves, encompassing both operations over the generic information model and the actual infrastructure, we repro-

 ISBN: 978-1-61208-271-4

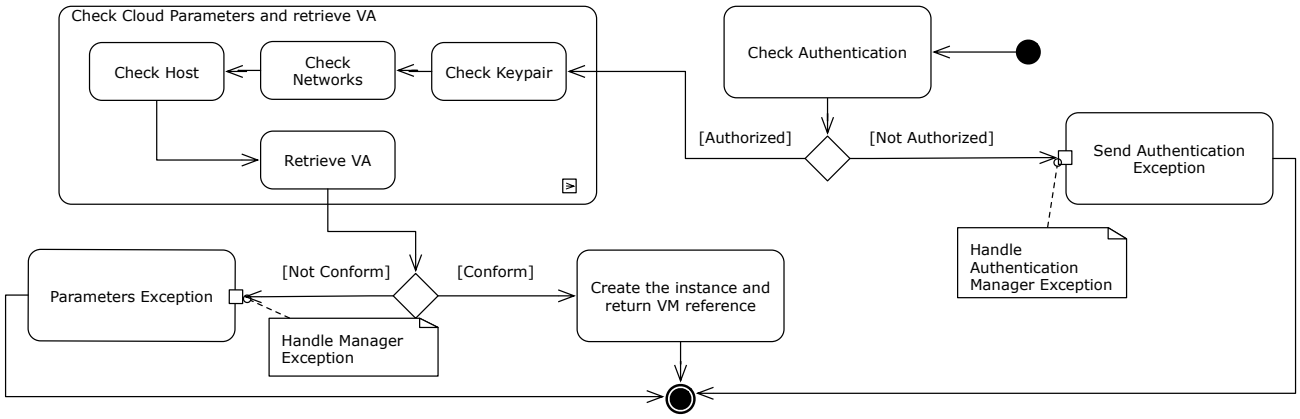

Figure 3. Starting a Virtual Machine

duce here a sample activity, the process of starting a VM (see Figure 3). This activity involves an authentication check (to be sure that the tenant can perform the creation), a state of the environment check (to ensure that the intended action is feasible), and the start of the VM action itself.

To complement our work with OpenStack and have a complete and usable management system, we have also developed two *Control Interfaces*: a REST service and a web page. The latter is intended to used by a human operator and the former is connected to an autonomic system of our own creation. The whole system was written in Java.

V. Conclusion and Future Work

In this paper, we have established the need for a management architecture for private IaaS clouds that support several solutions (to avoid data and vendor lock-in), working alone or together, and several user interfaces. To this end, we have proposed the use of a generic information model that captures all the relevant information for the infrastructure, and a modular architecture that can be adapted to fit several IaaS products and needs. We have detailed this architecture, making a special emphasis in how it achieves the desired results. In doing that, we have explained its three layers and how they fit inside the big picture.

To validate our approach, we have developed and tested a sample implementation with support for one cloud solution (OpenStack) and two interfaces (REST services and a web page). In this text, we have explained the aspects of this implementation more relevant to the development of the modules needed to support other cloud technologies.

In this process, we had to confront the realities of actual products and how to apply our proposal to them. This gave us some interesting realizations, like the pressing need for a more fine-grained network configuration support in clouds (already established in the literature [19]), and how to use the improved network customization features offered by solutions like Quantum to achieve this end.

Therefore, our next efforts will be focused on this topic. In the future, we also want to develop support for at least another cloud solution: This way we will be more able to test a federation of several private clouds and achieve a true working common management interface for multiple technologies. This matter will include the difficult topic of deciding where to physically put the management interface itself when working with several infrastructures. This integration will definitely test if our generic approach to security is suitable for use with different cloud solutions at the same time.

Finally, another line of work we want to follow is the application of our architecture to the management of public cloud offerings, since the interest in hybrid clouds that mix public and private IaaS is steadily growing.

References

[1] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, “Virtual infrastructure management in private and hybrid clouds,” Internet Computing, IEEE, vol. 13, no. 5, sept.-oct. 2009, pp. 14–22.

[2] N. Leavitt, “Is cloud computing really ready for prime time,” Computer, vol. 42, no. 1, 2009, pp. 15 –20.

[3] M. Armbrust et al., “A view of cloud computing,” Commun. ACM, vol. 53, no. 4, Apr. 2010, pp. 50 –58.

[4] A. Lonea, D. Popescu, and O. Prostean, “A survey of management interfaces for eucalyptus cloud,” in Applied Computational Intelligence and Informatics (SACI), 2012 7th IEEE International Symposium on, may 2012, pp. 261 –266.

[5] S. Wind, “Open source cloud computing management platforms: Introduction, comparison, and recommendations for implementation,” in Open Systems (ICOS), 2011 IEEE Conference on, sept. 2011, pp. 175 –179.

[6] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: Openstack and opennebula," in Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, may 2012, pp. 2457 –2461.

[7] L. Xu and J. Yang, "A management platform for eucalyptus-based iaas," in Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, sept. 2011, pp. 193 –197.

[8] C. Baun, M. Kunze, and V. Mauch, "The koala cloud manager: Cloud service management the easy way," in Cloud Computing (CLOUD), 2011 IEEE International Conference on, july 2011, pp. 744 –745.

[9] C. Baun and M. Kunze, "The KOALA cloud management service: a modern approach for cloud infrastructure management," in Proceedings of the First International Workshop on Cloud Computing Platforms, ser. CloudCP '11. New York, NY, USA: ACM, 2011, p. 1:1–1:6.

[10] "Scalr," http://code.google.com/p/scalr/, retrieved: 28th March 2013. [Online]. Available: http://code.google.com/p/scalr/

[11] "Puppet labs: IT automation software for system administrators," http://puppetlabs.com/, retrieved: 28th March 2013. [Online]. Available: http://puppetlabs.com/

[12] "Apache libcloud," http://libcloud.apache.org/, retrieved: 28th March 2013. [Online]. Available: http://libcloud.apache.org/

[13] "jcloud," http://www.jclouds.org/, retrieved: 28th March 2013. [Online]. Available: http://www.jclouds.org/

[14] "Apache deltacloud," http://deltacloud.apache.org/, retrieved: 28th March 2013. [Online]. Available: http://deltacloud.apache.org/

[15] H. Han et al., "A restful approach to the management of cloud infrastructure," in Cloud Computing, 2009. CLOUD '09. IEEE International Conference on, sept. 2009, pp. 139 –142.

[16] Z. Lu, J. Wu, and W. Fu, "A novel cloud-oriented ws-management-based resource management model," in Web Services (ICWS), 2010 IEEE International Conference on, july 2010, pp. 676 –677.

[17] T. Harmer, P. Wright, C. Cunningham, J. Hawkins, and R. Perrott, "An application-centric model for cloud management," in Services (SERVICES-1), 2010 6th World Congress on, july 2010, pp. 439 –446.

[18] R. García-Carmona, F. Cuadrado, A. Navas, A. Celorio, and J. Dueñas, "Multi-level monitoring approach for the dynamic management of private iaas platforms," Journal of Internet Technology, vol. Special Issue on Dynamic Intelligence for Sustainable Computing, no. to appear, 2013, p. to appear.

[19] J. Wickboldt, L. Granville, F. Schneider, D. Dudkowski, and M. Brunner, "A new approach to the design of flexible cloud management platforms," in Network and Service Management (CNSM), 2012 8th International Conference on, oct. 2012, pp. 155 –158.

 ISBN: 978-1-61208-271-4