Noname manuscript No.
(will be inserted by the editor)

Dynamic management of virtual infrastructures

Miguel Caballer · Ignacio Blanquer · Germán Moltó · Carlos de Alfonso

Received: date / Accepted: date

Abstract Cloud infrastructures are becoming an appropriate solution to address the computational needs of scientific applications. However, the use of public or on-premises Infrastructure as a Service (IaaS) clouds requires users to have non-trivial system administration skills. Resource provisioning systems provide facilities to choose the most suitable Virtual Machine Images (VMI) and basic configuration of multiple instances and subnetworks. Other tasks such as the configuration of cluster services, computational frameworks or specific applications are not trivial on the cloud, and normally users have to manually select the VMI that best fits, including undesired additional services and software packages. This paper presents a set of components that ease the access and the usability of IaaS clouds by automating the VMI selection, deployment, configuration, software installation, monitoring and update of Virtual Appliances. It supports APIs from a large number of virtual platforms, making user applications cloud-agnostic. In addition it integrates a contextualization system to enable the installation and configuration of all the user required applications providing the user with a fully functional infrastructure. Therefore, golden VMIs and configuration recipes can be easily reused across different deployments. Moreover, the contextualization agent included in the framework supports horizontal (increase/decrease the number of resources) and vertical (increase/decrease resources within a running Virtual Machine) by properly reconfiguring the software installed, considering the configuration of the

Miguel Caballer(✉) · Ignacio Blanquer · Germán Moltó · Carlos de Alfonso
Instituto de Instrumentación para Imagen Molecular (I3M). Centro mixto CSIC – Universitat Politècnica de València – CIEMAT Camino de Vera s/n, 46022 Valencia, Spain.
E-mail: micafer1@upv.es

Ignacio Blanquer
E-mail: iblanque@dsic.upv.es

Germán Moltó
E-mail: gmolto@dsic.upv.es

Carlos de Alfonso
E-mail: caralla@upv.es

multiple resources running. This paves the way for automatic virtual infrastructure deployment, customization and elastic modification at runtime for IaaS clouds.

Keywords Cloud Computing · Virtual Infrastructures · Contextualization

1 Introduction

With the advent of virtualization technologies and cloud infrastructures, scientists are exploring the usage of computational clouds for their research. Cloud computing technologies can offer: "ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources" [28]. Many scientific infrastructure providers are including IaaS as an added value [16] and [5] analyzes the use cases that scientific cloud infrastructures should cover. One of the most important features of cloud technologies and virtualization is that the software requirements are defined by the user rather than by the infrastructure provider. Moreover, to some extent, users can also define part of the hardware requirements. This is an important point to ease the migration of scientific applications into the cloud. However, the user is still required to prepare, deploy, configure and run the virtual appliances and the applications inside, requiring in some cases non trivial system administration skills. Nowadays there are different cloud providers, different software packages to deploy cloud platforms, etc. which makes even more difficult to integrate these technologies. Users need an easy way to define dependencies and application restrictions and delegate on a platform to automatically deploy, configure and monitor their virtual infrastructures.

This paper presents a set of components developed to simplify the automatic deployment and configuration of virtual infrastructures. The platform developed considers all the aspects related to the creation and management of virtual infrastructures: i) The software and hardware requirements specification for the user applications, using a simple language defined to be easy to understand by non-advanced users who just want to deploy a basic virtual infrastructure, but with enough expressivity for advanced users to set all the configuration parameters needed to get the infrastructure fully configured. ii) The selection of the most suitable Virtual Machine Images (VMI) based on the user expressed requirements. iii) The provision of Virtual Machines on the cloud providers available to the user, including both public IaaS Clouds (Amazon Web Services, Windows Azure IaaS, etc.) and on-premise resource provisioning systems (OpenNebula, OpenStack, etc.). iv) The contextualization of the infrastructure at run-time by installing and configuring all the required software that may not be available in the golden images (VMIs) used and, finally, v) the elasticity management, both horizontal (adding/removing nodes) and vertical (growing/shrinking the capacity of nodes). Considering the highly dynamic Cloud landscape, the platform should be extensible to future back-ends or standards.

In a previous work [1] an early prototype of the platform along with initial tests cases was presented. This paper describes an evolution of the platform featuring the following contributions: the contextualization phase, improving the functionality by enabling the deployment of a fully functional configured infrastructure, and the elasticity management. The automatic configuration and elasticity management at runtime are key issues in Cloud infrastructures and represent a major step forward with respect to our early prototype.

The remainder of the paper is structured as follows. First, section 2 analyzes the current tools for virtual infrastructure deployment. Then, section 3 describes the software architecture of the Infrastructure Manager along with the underlying components used. Then, section 4 describes a case study to test the suitability of this platform for the deployment of virtual clusters. Finally, section 5 summarises the paper and points to future work.

2 State of the art

There exist works in the literature and software tools that address the deployment of virtual infrastructures. In the next paragraphs different approaches are analyzed, detailing the achievement of the objectives described in the introduction. Finally Table 1 shows a comparison of the features of all the analyzed works.

Several cloud providers, such as Amazon Web Services (AWS), provide tools to deploy virtual infrastructures. In particular, CloudFormation [3] provides developers and systems administrators with an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable fashion. It provides tools to launch a set of VMs in a coordinated fashion, and can also configure a set of Amazon services (Elastic Load Balancer, Auto Scaling, etc.). AWS CloudFormation introduces two concepts: *Template*, a JSON text-based file that describes all the AWS resources you need to deploy and run your application; and *Stack*, the set of AWS resources that are created and managed as a single unit when a template is instantiated by AWS CloudFormation. The user must select the image of the VM from the Amazon Machine Image (AMI) catalog. This catalog only provides very basic information about the AMIs (architecture, O.S. and a free text field with a description), so the user must previously know the configuration and software installed in the image to use. This is an important problem that hinders reusing existing VMIs. Other limitation of this tool is that it can only be used in the Amazon EC2 infrastructure.

The Nimbus project team group has developed a Context Broker [21]. This context broker enables the contextualization of VMs to create the so-called "One-Click Virtual Clusters". A created set of VMs are configured according to some roles inside the cluster distribution. The VMs are launched using the Nimbus commands, so it can only be used in Nimbus-based cloud providers, or Amazon EC2 using the "IaaS Gateway" component. It also requires that the user chooses the required VMI. In addition, the contextualization has some limitations, such as the use of simple scripts that must be stored in the VMIs, so the images must be specially customized for each application. Furthermore, the tool does not offer recontextualization of the infrastructure if new nodes are added, thus hindering the elasticity management. The same research group has developed other system [22, 26] that enables extending a "real" cluster elastically adding new resources over a cloud infrastructure (with cloud bursting techniques). In this case, to avoid the reconfiguration limitations of the Context Broker they use a combination of Chef [19] and a new developed component called Recontextualization Broker to perform the initial configuration and later reconfiguration of the cluster when new nodes are added or removed. This new component solves some of the problems of the previous software but it cannot be used as a general solution since it focuses on the deployment of HPC clusters and cannot deploy other virtual infrastructures. In a further

work, the same group has developed another tool called cloudinit.d [6] designed to launch, control and monitor cloud applications. It automates the VM creation process, the contextualization and the coordination of service deployment. It supports multiple clouds and the synchronization of different "runlevels" to launch services in a defined order. Furthermore it provides a system to monitor the services that uses user-created scripts to ensure that they are running. This system checks for service errors, re-launching failed services or launching new VMs. However, the static selection of VMIs is still required. It enables the contextualization of VMs using simple scripts, which are insufficient in scenarios with multiple VMs with different Operating Systems.

Claudia [30, 37, 38] enables the deployment of a set of VMs in IaaS environments, using an extension of the Open Virtualization Format (OVF) [14] called Service Description File (SDF). Static selection of VMIs are used in the deployment stage, and no contextualization is made, so it is based in the software previously installed in the VMIs. The SDF language includes an elasticity section adding rules to manage the evolution of the size of the cluster using the defined "Key Performance Indicators" (KPI). It has a modular design with a plug-in system which can be used to add new deployment types. It currently supports only OpenNebula.

Apache Whirr [4] supports deploying clusters both on EC2 and Rackspace. The user specifies the number of instances needed and the roles they must provide. It has the same problem of the static selection of VMIs. It has been initially designed to launch hadoop clusters, but it can be extended adding new Java classes to implement the installation and contextualization of the new roles defined. It does not enable elasticity management, so once the cluster is launched its size cannot be modified by the user.

Wrangler [20] has a specific orientation to launch VMs in a general way. It enables the users to define their requirements using an XML file. In this file the user can also specify the scripts to be used to configure the VMs to adopt a specific role within the virtual infrastructure. In this case the scripts are stored in the wrangler coordinator node, so it does not need to be previously copied in the VMIs. But the VMIs indeed require a prior step to prepare them by installing the wrangler agent and configuring it to connect to the coordinator node.

SixSq SlipStream[1] provides a web portal to deploy and configure a set of VMs. The configuration of the nodes are made by means of a list of packages to install and a script file that can be executed in each VM. Each script is completed with a set of parameter values as "hostname" or "instanceid" to enable to create more functional and customized scripts. It enables to access a large list of Cloud platforms, both public and on-premise: EC2, Azure, OpenNebula, OpenStack, OCCI, etc. The main limitation is the static selection of VMIs. In addition, it does not enable elasticity management and, therefore, once the infrastructure is deployed its size cannot be modified by the user.

Vagrant [18] was initially designed to launch VMs over the VirtualBox virtualization platform but it has a modular design that enables new providers to be added. Currently it has support to VirtualBox, VMware Fusion and EC2. It enables the management of a set of VMs using the "multi-machine" environments. It also supports the usage of provisioning tools such as shell scripts, Chef, or Puppet to automatically install and configure software on the machines. As in the previous

[1] https://slipstream.sixsq.com

Table 1 Virtual Infrastructure deployment tools comparison.

	Wrangler	Whirr	Claudia	Cloudinit.d	SlipStream	Recont. Broker	Cloud Formation	Vagrant	TOSCA
Systems support	EC2, Eucalyptus, OpenNebula	EC2	OpenNebula	EC2, Nimbus	EC2, OpenStack, OpenNebula Azure, ...	EC2, Nimbus	EC2	VirtualBox, VMWare, EC2 EC2	-
VMs Context.	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes
High level Context. Lang.	No	No	-	No	No	Yes	-	Yes	No
VM pre-config.	Yes	Yes	Yes	Yes	No	No	No	Yes	No
Simple deploy lang.	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No
Extensible	Yes	Yes	Yes	No	No	No	No	Yes	Yes
Catalog of VMIs	No	No	No	No	No	No	No	No	-
Elasticity Mgmt	Yes	No	Yes	No	No	Yes	Yes[1]	Yes	Yes

case to use any of these tools they must be installed previously in the VMIs to use. It provides the "box" concept to encapsulate the VMIs for each provider. A "box" is a tar file where all the files needed to encapsulate a VMI in the Vagrant environment. In the case of VirtualBox and VMware this box file contains the image itself. In the case of EC2, it contains a reference to the AMI. However, Vagrant is designed to work in a single machine and, therefore, it does not support the deployment of large virtual infrastructures.

Currently the are also some initiatives from standardization organizations, such as OASIS with the Topology and Orchestration Specification for Cloud Applications (TOSCA) [31] to describe service templates across *aaS layers and connecting them to the resource abstraction layer. It enables the description of a service with a high level topology and plan for implementation and configuration. A service specified with TOSCA typically describes virtual hardware, software, configuration, and policies necessary to orchestrate the service. Currently it provides no implementation, but some cloud software packages as OpenStack is studying the possibility of the integration of TOSCA with their software stack. Although this specification considers the contextualization of the VMs it does it using executable files, and not using some high level contextualization language.

One common limitation in all the analyzed systems is the usage of manually selected base images to launch the VMs. It is an important limitation because it implies that users must create their own images or they must previously know the details about software and configuration of the image selected. This limitation affects the reutilization of the previously created VMIs, forcing the user to waste time testing the existing images or creating new ones (as an example in Amazon EC2 there are thousands of AMIs). Another issue is that most of them need to use a VMI specifically prepared for their tools, requiring a specific software installed, a set of scripts prepared, etc. Another important limitation is the usage of simple scripts in the contextualization, instead of higher level languages such as Puppet [35], Chef [33], Ansible [13], etc. which enable the creation of system independent configurations. Only the Nimbus "Recontextualization Broker" uses Chef to perform these tasks. This problem is exacerbated in some tools where the configuration scripts must also be stored in the base image of the VM.

[1] Using the Auto Scaling service from Amazon.

3 Architecture

In order to support all the functionality issues addressed in Table 1, an architecture has been developed, which is depicted in Figure 1. The different components are described.

The first component of the architecture is the language describing the requirements of the virtual infrastructure, called the Resource and Application Description Language (RADL). This language has been designed to be simple for non-advanced users, who can deploy infrastructures by specifying very basic information. Additionally, savvy users can access more advanced functionality. The Virtual Machine Image Repository & Catalog (VMRC) enables indexing and storing the VMIs including all the relevant information about them, including the software applications installed. The Infrastructure Manager (IM) implements a service that provides APIs, using standard connection methods (XML-RCP, and REST), with a relatively simple set of functions to provide the functionality required in the management of VMs. It is also in charge of orchestrating the deployment of virtual infrastructures using the rest of components. The last component is the contextualization system, which installs and configures all the software not yet available in the VMIs from the VMRC and described in the RADL. It also deals with the reconfiguration in the case of adding or removing nodes. This system uses Ansible [13] integrated with a set of developed tools to build up the contextualization system. Finally, client tools (command line and web application) have been developed to ease the access to the functionality of this platform for the user.

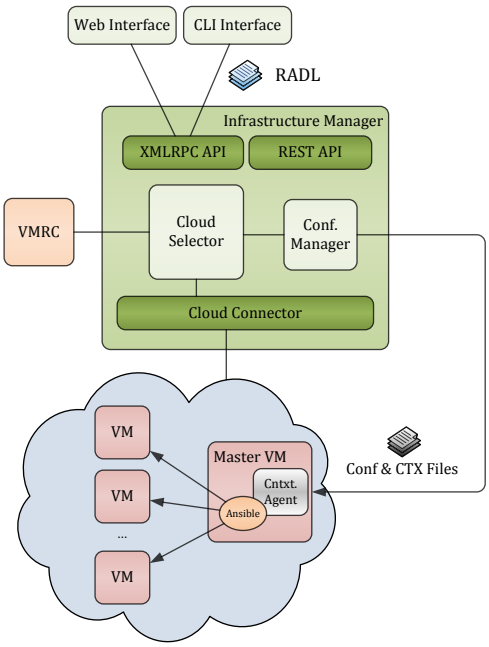

Fig. 1 Infrastructure Manager Architecture

The following subsections describe the aforementioned components.

3.1 Resource and Application description Language (RADL)

The main purpose of the Resource and Application description Language (RADL) is to specify the requirements of the resources where the scientific applications will be executed. It must address not only hardware (CPU number, CPU architecture, RAM size, etc.) but also software requirements (applications, libraries, data base systems, etc.). It should include all the configuration details needed to get a fully functional and configured VM (a Virtual Appliance or VA). It merges the definitions of specifications, such as OVF, but using a declarative scheme, with contextualization languages such as Ansible. It also allows describing the underlying network capabilities required.

The initial scheme of the RADL language is described in [1], where a minimal example is shown in Figure 2. This section presents a brief description of the RADL scheme focusing on the new contributions. A typical RADL file includes the following sections:

- Environment features. This includes devices, services, etc. not provided by the VMs, but requiring some interaction from the VMs. Some examples are networks, Windows Active Directory Service, Storage Area Networks, etc. Currently only networks are implemented and they are assumed to be Local Area Networks (LAN) that the VMs can use to connect to the other VMs and to other external networks. This part is defined with the reserved word "network".
- Type of nodes definition: The reserved word "system" is used to specify the section to describe all the features (hardware and software) of one type or role of nodes in the infrastructure. A role describes a group of nodes that have the same features.
- Deploy instructions: A set of deploy instructions to specify the number of instances of the defined "system" types to effectively launch. Two additional parameters can by specified (not shown in the example): The first one enables specifying the cloud platform on which to deploy the VMs. The valued specified must be one of the IDs defined in the authorization data (see section 3.3.1). The second one enables specifying a deployment priority. The instances with a higher number will be deployed later to the nodes with a lower number, to create an ordered deployment. It enables, for example the deployment of a database server before deploying an application server that needs to have the connection to the database active to start functioning.
- Configuration section: The reserved word "configure" is used to specify the section where the user can specify (using Ansible's language) the necessary recipes to configure the VMs. Each "configure" section is defined with a name and can include as many configure sections as needed. In addition, one configure section can "include" another, enabling to reuse existing configure sections. Finally, for each instance of type defined in the "system" section, only the instructions of the corresponding configuration section will be applied. These features will be clarified in the examples.

These last two features described (ordered deployment and configuration section) have been added in the new release of the RADL language.

```
system nodeS (
  memory.size>=1024M and
  disk.0.applications contains (name='tomcat')
)

deploy nodeS 2
```

Fig. 2 Example of a simple RADL file

```
network red (outbound = yes)

system nodeB (
system='kvm' and
cpu.count>=1 and cpu.arch='i686' and
memory.size>=1024M and
net_interface.0.connection='red' and
net_interface.0.dns_name='node-#N#' and
disk.0.os.name='linux' and
disk.0.os.flavour='ubuntu' and
disk.0.os.version>='9.10' and
disk.0.applications contains (name='tomcat')
)

configure common (
@begin
---
  - tasks:
    - user: name=user1 password=a7ae2ax1k0a
@end
)

configure nodeB (
@begin
---
  - tasks:
    - include: common.yml
    - yum: pkg=${item} state=installed
      with_items:
      - torque-client
      - torque-server
@end
)

deploy nodeB 10
```

Fig. 3 Example of a extended RADL file

The RADL sample in Figure 2 shows how a non-advanced user can define a virtual infrastructure only by specifying the application relevant data. In particular the example requires two VMs with at least 1GB of RAM with the Apache Tomcat application server installed. The rest of parameters required to finally deploy the VM are generated by the platform, using default values (i.e. one 32-bit CPU, one network connection with outbound connectivity, etc.).

Figure 3 shows another example of a complete RADL document that includes most of the advanced features of the RADL language. In the top of the example the basic VM characteristics are shown: An x86 32-bit CPU with 1 GB of RAM and

an outbound network connection, Ubuntu 9.10 (or higher) as the OS and Apache Tomcat installed. The "dns_name" field has been added to the network interfaces enabling the user to specify the DNS name to the interface. The name used in the interface with the id "0" will be used as the name of the host. It is important to notice that all the DNS names will only be visible for the nodes included in the infrastructure. This field must be carefully used in case of deploying a set of VMs using the same "system" definition, because all the VMs will have the same DNS name. The substitution string "#N#" has been added to deal with this issue. This string will be replaced with the number of the instance inside the infrastructure. For example if a DNS name like "node-#N#" is used, the name of the VMs will be node-0, node-1, etc.

The initial version of the language [1] had limited functionality because it did not include support to define the tasks that can be automated to configure a virtual infrastructure. The new "configure" section uses Ansible's language to specify all the configuration details. In the example two "configure" sections appear: common and nodeB. The former does not match with any "system" defined, so it will not be applied, but it will be available to be used by the nodes using the Ansible "include" statement. The latter will be applied to all the VMs of the "nodeB" type. As this one includes the "common configure", all the nodeB VMs will create a user named "user1" and then install the torque client and server packages, as is defined in the configure section.

The Infrastructure Manager (IM) creates a set of variables to enable some information of the RADL language to be accessible from Ansible. These variables are very useful to perform some configuration tasks, as using IM_MASTER_FQDN to set the name of the front-end node in the configuration process of any client-server system, identifying the current node ID or HOSTNAME to download the correct files to a specific VM, etc.

- IM_NODE_HOSTNAME: Hostname of the node being processed (without the domain).
- IM_NODE_FQDN: Complete FQDN of the node being processed.
- IM_NODE_DOMAIN: Domain of the node being processed.
- IM_NODE_NUM: Number of the instance of the node being processed.
- IM_MASTER_HOSTNAME: Hostname of the master node (without the domain).
- IM_MASTER_FQDN: Complete FQDN of the master node.
- IM_MASTER_DOMAIN: Domain of the master node.

For each of the applications pre-installed, and added as metadata in the VMRC, in the VMI used, the IM also provides the following variables:

- IM_[application name]_VERSION: Version of the application [application name].
- IM_[application name]_PATH: Install path of the application [application name].

3.2 Virtual Machine image Repository and Catalog (VMRC)

The VMRC (Virtual Machine image Repository and Catalog) [11] is used to find a suitable Virtual Machine Image (VMI) that satisfies the requirements of the user (in terms of Operating System, CPU architecture, applications installed, etc.), and

it is compatible with the hypervisor available in the Cloud system. This component is used to index the VMIs that are typically stored in the different Cloud deployments, so that they can be reused in multiple contexts. It also implements matchmaking algorithms to obtain a ranked list of VMIs that satisfy the aforementioned given set of requirements. Depending on the scenario, the VMI descriptions can be created only by the administrator of the VMRC, with curated and tested images, or by a set of users responsible for creating and maintaining the virtual machines required for specific virtual organizations, or sites.

An URI naming convention has been defined to enable the registration of VMIs in the VMRC when the VMIs are stored in the cloud providers. This URIs enable the IM to know their location and how to access them. The protocol field of the URI is used to specify the cloud provider type: one (OpenNebula), ec2 (Amazon EC2) and ost (OpenStack). In the case of OpenNebula and OpenStack, the address and port fields have their default function and the path is used to specify the ID of the images, an integer number for OpenNebula and a string for OpenStack. In the case of EC2, the address field is used to specify the region where the image is stored and the path to store the name of the AMI.

- one://server:port/image-id
- ost://server:port/ami-id
- ec2://region/ami-id

3.3 Infrastructure Manager - IM

The main goal of the Infrastructure Manager is to provide a set of functions for the effective deployment of all the required virtual infrastructures needed to launch an application in a cloud environment and then managing them on demand during all the execution time.

The IM provides two sets of APIs to enable high-level components to access the functionality. The first APIs uses the XML-RCP protocol, that can be called the "native" API. Second, a REST API has been implemented, given its increasing popularity. There are also secured versions of both APIs using Secure Sockets Layer (SSL) to encrypt the communications. These APIs provide a set of simple functions for clients to create, destroy, and get information about the infrastructures. The RADL language is used both to create and to get the information about the infrastructures. The IM also provides functions to add and remove resources and modify the features of the existing ones, both hardware and software at run-time.

Figure 1 shows the architecture of the Infrastructure Manager. On the top, the client interfaces currently available for users are depicted. The IM in the center of the figure provides the upper layers with the functionality through the APIs provided. The IM uses the "Cloud Selector" component to query the VMRC service for the list of VMIs that best fit the user requirements (expressed in the RADL document) and merge this information with the list of available cloud providers for the user, in order to get the best option. The "Cloud Connector" layer is used to provision the VMs in the cloud providers. It provides an homogeneous interface to connect with the different cloud middlewares. Finally, once the VMs are deployed and in the running state, the "Configuration Manager" is in charge of managing the contextualization of all the VMs of the infrastructures using the Ansible tool.

```
id = one; type = OpenNebula; host = server:2633; username = user; password = pass
id = one2; type = OpenNebula; host = server2:2633; username = user2; password = pass2
id = libvirt; type = LibVirt; host = server; username = user; password = pass
type = InfrastructureManager; username = user; password = pass
type = VMRC; username = user; password = pass
id = ec2; type = EC2; user = id; password = key
id = ost; type = OpenStack; host = server:8773; username = id; password = key
id = occi; type = OCCI; host = server:4567; username = user; password = pass
```

Fig. 4 Authorization data examples

3.3.1 Authorization Data

The authorization data is used to validate access to the components in the infrastructure. Therefore, it must be included in all the calls to the APIs. The native API requires including this authorization data as the last parameter in every call. The REST API requires these data to be placed inside the "AUTHORIZATION" HTTP header. This parameter is composed of a set of "key - value" pairs, where the user specifies the authorization data for all the components and cloud providers available. Figure 4 shows examples of authorization data.

The list of "key" values that must be specified for each component are:

- id: An optional field used to identify the virtual deployment. It must be unique in the authorization data.
- type: The type of the component. It can be any of the components of the architecture, such as the "InfrastructureManager", "VMRC" or any of the cloud providers currently supported by the IM: OpenNebula, EC2, OpenStack, OCCI or LibVirt.
- username: The name of the user for the authentication. In the EC2 and OpenStack cases it refers to the Access Key ID value.
- password: The password for the authentication. In the EC2 and OpenStack cases it refers to the Secret Access Key value.
- host: The address to the server in format "address:port" to specify the cloud provider to access. In the EC2 and in the system components (IM and VMRC) this field is not used.

Using the authentication data the "Cloud Selector" will get the list of available cloud providers for the user. This list may be different in each system call. It provides more flexibility to the system, thus avoiding the maintenance of a list of fixed cloud providers.

3.3.2 Cloud Selector

The Cloud Selector (CS) is the central component of the IM. It will select the best combination of the available VMIs and cloud providers. To perform this task, it must contact the VMRC to select the VMIs that best fits the user's requirements, in terms of operating systems and software installed. Then, it will use the user credentials to get list of available cloud providers. The CS must select the cloud providers compatible with the VMIs obtained from the catalog. Reciprocally, it

must select the VMIs with the appropriate format to be launched in the cloud provider selected, to finally combine the information to get the best "cloud - VMI" pair.

The selection of the "best" provider is not trivial. There are many factors to consider like the number of total and available resources, performance of the VMs, price, location, etc. Moreover, most of the cloud providers do not provide information about the underlying infrastructure such as the total number of resources, available resources, etc. This case is even more complicated, since the IM can access not only cloud providers like Amazon EC2, where you know the theoretical performance of the VMs, but also OpenNebula deployments or LibVirt virtualization platforms, which do not provide this information. For public Cloud platform selection there are some interesting works: STRATOS [34] facilitates the deployment of Cloud application topologies from multiple Cloud providers using cost as the main objective, by means of multi-criteria optimization algorithms. CompatibleOne [41] considers not only the user specified cloud provider constraints but also a wide range of objectives as financial, energetic, geographic or operator contractual preferences to select the best cloud provider. Other works as [8, 12, 24, 36, 39] describe several solutions to select the best cloud provider using SLAs or K-nearest neighbour algorithms. Other simple solution is to select the cheapest provider, but this cannot be applied in private or on-premise providers.

In our case, the CS will first select the most suitable VMI located in the cloud providers specified in the authentication data. It will use the *Suitability Value* (SV) returned by the VMRC [11] that considers the soft weights specified by the user in the RADL applications requirements. This means that the VMI that best satisfies the requirements and preferences of the user will receive a higher SV. Then it will select the cloud provider where the image is located to launch the VM. If some images obtain the same SV, the CS uses the order specified by the user in the authentication data to select the image - cloud provider pair.

VM co-location dependencies that force a set of them to be launched in the same cloud deployment (infrastructure) must also be considered. The most common one is that a set of VMs has a common private network. In this case the CS will select the pair image - cloud provider for this group forcing that all of them are in the same cloud provider. The usage of VPNs will be studied in the future to remove this restriction.

3.3.3 Cloud Connector

Currently, a lot of different cloud providers are available. Most of them are using different and non-standard connection protocols. Although the proprietary protocols used by Amazon Web Services, being a pioneer and the largest IaaS provider, are becoming "de facto" standards, it is difficult to find an open standard to homogeneously access IaaS cloud infrastructures.

There are many works in the literature [7, 23, 25] regarding cloud interoperability or federation, because only through federation and interoperability can cloud providers take advantage of their aggregated capabilities to provide a seemingly infinite service computing utility. But these solutions are based on the usage of open standards.

Different initiatives have appeared in the last years to create an open standard API to access IaaS clouds to provide this vision of federated clouds: Open Cloud

Computing Interface (OCCI) [32] is an Open Grid Forum recommendation of a protocol and a REST API designed to access IaaS clouds. Cloud Infrastructure Management Interface (CIMI) [15] is a standard proposed by the Distributed Management Task Force (DMTF) to simplify the cloud infrastructures management. TCloud API Specification [40] is a cloud API proposed by Telefonica based in the vCloud 0.8 API published by VMware. Although OCCI is the most extended standard with the largest number of implementations, not all the implementations of OCCI are fully compatible, thus making the interoperability difficult.

To address these incompatibility issues, some "aggregation APIs" have appeared. Apache Libcloud[3] and Deltacloud[4] are the most widely used. Although these tools are very useful in some scenarios, there are some issues that complicate their usage. For example, Libcloud does not have all the basic functionality (launch and terminate VMs) for all of its "drivers". Moreover, the support of OpenNebula (the main cloud software used in our private cloud) is very restricted through the OCCI interface, and the native interface is not supported. Deltacloud has the same problems accessing OpenNebula deployments. In addition, in order to use the REST API, you should launch as many servers as the number of different cloud providers, which complicates their deployment and usage.

Due to the difficulty to find an homogeneous way to access the different cloud providers, the IM has added a new abstraction layer to enable the interoperability with different IaaS clouds, until a real open standard is defined and used widely. This layer has been designed with a simple API with 6 functions to provide the basic functionality needed by the IM:

- Launch a VM: Create and run a VM following the requirements defined in the RADL.
- Terminate a VM: Completely terminate the VM.
- Get VM information: Get the VM information in RADL format.
- Stop a VM: Stop (but not destroy) a VM.
- Resume a VM: Start a previously stopped VM.
- Modify VM: Modify the properties of a running VM, to provide vertical elasticity functionality where the underlying platform supports this feature.

This layer has been designed using a plug-in scheme to ease its extension. We currently provide plug-ins for: OpenNebula, OCCI, Amazon EC2, OpenStack and libvirt. This set of plug-ins enables access to a large number of cloud providers and virtualization platforms, thus enabling the user to start using a simple virtualization system and then transparently migrate to the cloud.

3.3.4 Configuration Manager

The management of the contextualization process in the infrastructure is orchestrated by the Configuration Manager (CM) component. It is in charge of managing all the steps to perform the configuration tasks needed for a fully functional infrastructure, considering the user requirements and using the appropriate tools.

There are two options when it comes to perform the contextualization. The first one is to require the VMIs to have the contextualizator (or configuration agent)

[3] `http://libcloud.apache.org/about.html`

[4] `http://deltacloud.apache.org/about.html`

pre-installed. This option eases the complexity of the CM, but it complicates the reutilization of the VMIs since they must be previously customized to the deployment environment. Moreover, as these tools usually uses a pull scheme, using a client-server approach, it is necessary to configure the clients with a fixed address to connect to the server. These kind of solutions are widely extended in platforms such as EC2[5] and other deployments with the same interface as OpenStack[6] (using the 169.254.169.254 IP)

The second option is to use a set of basic scripts to install and configure the contextualizator during the infrastructure launch process. This second option is more flexible since the VMIs do not require a proper customization beforehand, but notice that this involves an extra step in the infrastructure deployment process. This has been chosen for the proposed platform. One of the steps performed by the CM is the installation and configuration of the contextualizator in the deployed infrastructure, thus being able to interact with any VMI.

As stated in previous sections, there are many contextualization tools such as Puppet [35], Chef [33], or Ansible [13]. The latter has been selected over other contextualization tools since it only requires SSH-based access on the machines being configured (thus avoiding to have pre-packaged VMIs). In addition, it uses a "push" approach, enabling one node to have the control of the contextualization process and precisely know when a certain configuration has been applied to a given machine.

The CM is launched by the IM once the infrastructure has been created in the cloud provider. Then the steps performed by the CM to contextualize the infrastructure are the following:

1. Get the list of applications to install from the user RADL and compare them with the list of pre-installed applications in the VMIs, obtained from the VMRC. This will determine the list of applications to install.
2. Choose one of the VMs with public IP and wait until it is running and can be accessed via SSH. The credentials obtained from the VMRC are used to access the VMs. This VM is named as the "master" node and it is used to bootstrap the configuration of the other VMs.
3. Configure the "master" VM. This process implies installing and configuring Ansible, including copying all the necessary recipes to install the applications selected by the user and the configuration recipes included in the RADL.
4. The CM launches a contextualization agent that is in charge of coordinating all the contextualization tasks using the Ansible API. The fist step is to wait for all the VMs to have the SSH access active to finally call Ansible to configure all the VMs of the infrastructure (including the master VM itself).

3.3.5 Elasticity management

The elasticity management, as one of the key features of the cloud infrastructures, is an important task enabling the virtual infrastructures to adapt their resources to the dynamic requirements of the applications.

[5] `http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/AESDG-chapter-instancedata.html`

[6] `http://docs.openstack.org/trunk/openstack-compute/admin/content/metadata-service.html`

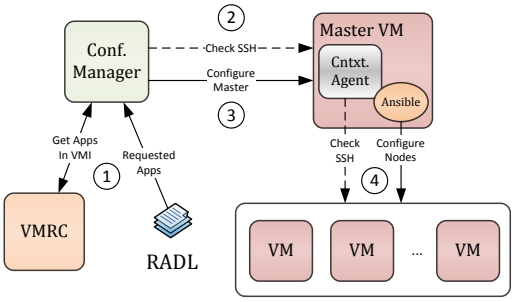

Fig. 5 Configuration Manager Steps

The IM provides functions in its API to support two models of elasticity management. On the one hand, horizontal elasticity enables adding or removing VMs to an infrastructure. On the other hand, the vertical elasticity enables an individual VM to adapt its features to the application requirements (mainly in terms of CPUs and RAM) [17]. The IM exposes this features to higher-layer frameworks which provide with the needed monitoring and decision-making systems to trigger the elasticity. As shown in the section on Use Cases 4, the EC3 layer [10] provides IM with this functionality.

Currently the horizontal elasticity management is easier as it is supported intrinsically in all the Virtual Machine Management (VMM) systems. The management of the vertical elasticity is more complex since it must be supported by all the virtualization elements: the VMM the hypervisor and the operating system of the VM. Most of the current operating systems (Windows, Linux, etc.) support this kind of feature using "memory ballooning" techniques. In addition, most of the hypervisors provide support (KVM, XEN, Hyper-V, VMWare, etc.). But there are no VMM that provides native support to this kind of features. A work to extend the OpenNebula API to support vertical elasticity on top of KVM platforms was developed by the authors as a proof of concept [29]. Furthermore it must be considered that, in some cases, adding new resources to a VM may trigger a VM migration to another physical node with enough resources. This process is done with the "live migration" process to avoid interruptions in the VM functionality.

The IM provides support to both elasticity techniques to enable other software layers that provide monitoring functions (as in [29]) to manage the elasticity of the infrastructures. Although the IM does not provide monitoring functions, it has been integrated with the Ganglia monitoring system [27] to show the monitoring information of the infrastructure merged with the common information provided by the IM in RADL format, adding new properties such as as: "disk.free", "cpu.usage", "memory.free", etc. If the deployed virtual infrastructure has installed and configured Ganglia this information will be shown in the IM information system. The IM provides contextualization facilities to enable the installation and configuration of Ganglia in the virtual infrastructure thus enabling non advanced users to get the monitoring information. In that case the user must add Ganglia as an application requirement in the RADL. In the "master" node adding the "gmetad" application and "ganglia" in the other nodes.

4 Use Cases

In this section two different use cases will be shown to demonstrate the functionality of the system as well as to show the time required for the infrastructure deployment using different types and sizes showing the versatility and scalability of the proposed system. Initially it will be shown a use case of the platform, from the point of view of a user that access directly to the functionality of the IM service to launch his virtual infrastructure. Then other services will be shown that use the IM API to create a more complex system with enhanced functionality.

4.1 Scientific infrastructure deployment - EMI Cluster

A representative use case is the set-up of a grid node based on the EMI-2[7] distribution. The use case will consist of deploying and configuring an EMI-2 cluster with Torque batch system. Grid nodes share their resources in the frame of the EGI initiative[8]. The workload in EGI is unpredictable and the capability to deploy and undeploy nodes of a cluster will be very effective. A virtual cluster will be launched expressing all the requirements in a RADL document (Figure 6). Then, the process of adding or removing nodes is analysed, measuring the time needed for each step, to show the elasticity capabilities of the platform.

The steps described in the "configure" sections of the RADL are the standard of the EMI software[10]. The user should create all the configuration files needed to configure the EMI software: site-info.def, groups.conf, users.conf and put them in an accessible URL. Or they can be also included in the recipe. In this case the first solution has been selected to create a shorter RADL document.

One of the advantages of the proposed system is that it enables the addition and removal of nodes with a simple call to the IM. It will reconfigure the whole infrastructure to continue working properly. The IM makes the management of clusters easier by adding or removing nodes on demand.

Three different tests has been made using two cloud providers. The first cloud provider is an on-premise cloud with OpenNebula composed of a set of three Dell blades (M600 and M610), each one with two Quad-Core processors and 16 GB of RAM. The second one uses the public cloud infrastructure of Amazon EC2. Initially a cluster with 6 nodes will be launched both in the OpenNebula and EC2 platforms. Finally a larger cluster with 32 nodes will be launched in the EC2 cloud. It is worth mentioning that the RADL document used in the two first cases is the same. The only change resides in the user credentials and in the last case only the number in the "deploy" instruction was changed.

The time needed to deploy the infrastructure can be decomposed into the following steps:

– VM Running: Time needed to have the master VM in the running state. This time is quite short in both cases. In the OpenNebula case the base VMI selected is relatively small thus requiring short time to transfer. In the EC2 case the AMI uses an EBS (Elastic Block Store) volume.

[7] `http://www.eu-emi.eu/`
[8] `http://www.egi.eu/`
[10] `https://twiki.cern.ch/twiki/bin/view/EMI/GenericInstallationConfigurationEMI2`

```
network publica (outbound = 'yes')
network privada

system front (
cpu.arch='x86_64' and
cpu.count>=1 and
memory.size>=1024m and
net_interfaces.count = 2 and
net_interface.0.connection = 'publica' and
net_interface.0.dns_name = 'pbsserver.i3m.upv.es' and
net_interface.1.connection = 'privada' and
disk.0.os.name='linux' and
disk.0.os.flavour='scientific' and
disk.0.os.version='6.3'
)
system wn (
cpu.arch='x86_64' and
cpu.count>=1 and
memory.size>=1024m and
net_interfaces.count = 1 and
net_interface.0.connection = 'privada' and
net_interface.0.dns_name = 'wn-#N#' and
disk.0.os.name='linux' and
disk.0.os.flavour='scientific' and
disk.0.os.version='6.3'
)

configure common (
@begin
- template: src=utils/templates/hosts.conf dest=/root/wn-list.conf
- get_url: url=http://.../EGI-trustanchors.repo dest=/etc/yum.repos.d/EGI-trustanchors.repo
- command: yum -y install http://.../epel-release-6-8.noarch.rpm creates=/etc/yum.repos.d/epel.repo
- command: yum -y install http://.../emi-release-2.0.0-1.sl6.noarch.rpm creates=/etc/yum.repos.d/emi2-base.repo

- yum: pkg=${item} state=installed
  with_items:
    - ca-policy-egi-core
    - emi-torque-client
    - emi-wn
- get_url: url=$RSCF/$item dest=/root/$item
  with_items:
    - site-info.def
    - groups.conf
    - users.conf
@end
)
configure front (
@begin
  - vars:
     ntp_server: ntp.upv.es
     RSCF: http://web.i3m.upv.es/RSCF/UMD
    tasks:
    - include: conf_common.yml
    - include: ntp/tasks/ntp.yml
    - yum: pkg=emi-torque-server state=installed
    - command: /usr/sbin/create-munge-key creates=/etc/munge/munge.key
    - file: path=/etc/munge/munge.key owner=munge group=munge mode=0400
    - service: name=munge state=started
    - command: /opt/glite/yaim/bin/yaim -c -s site-info.def -n TORQUE_client -n WN -n TORQUE_server chdir=/root
@end
)
configure wn (
@begin
  - vars:
     ntp_server: ntp.upv.es
     RSCF: http://web.i3m.upv.es/RSCF/UMD

    tasks:
    - include: conf_common.yml
    - include: ntp/tasks/ntp.yml
    - copy: src=/etc/munge/munge.key dest=/etc/munge/munge.key owner=munge group=munge mode=0400
    - service: name=munge state=started
    - command: /opt/glite/yaim/bin/yaim -c -s site-info.def -n TORQUE_client -n WN chdir=/root
@end
)

deploy front 1
deploy wn 1
```

Fig. 6 RADL document of the Use Case. Notice that long URLs have been shortened for the sake of clarity. Machine names have been anonymized

– VMs Accessible: Time needed to have the SSH port accessible in the master VM. This time depends on the time used by the Operating System (OS) of the VMs to boot and launch the SSH server.

Table 2 Virtual Infrastructure creation times

	6 node (ONE)	6 nodes (EC2)	32 nodes (EC2)
VM Running	0:31	1:02	0:33
VMs Accessible	2:12	0:52	0:56
Ansible Configured	2:49	3:39	3:09
System Configured	31:21	17:53	20:34
Total Creation	36:53	23:26	25:12
New VM running	0:40	0:50	1:01
Reconfiguration	25:02	12:09	12:35
Total Addition	25:42	12:59	13:36
VM Removed	0:03	0:03	0:04
Reconfiguration	2:20	2:11	2:15

– Ansible Configured: Time needed to install and configure the Ansible contextualization software in the master node. This is a relatively simple process that implies to download the software and a small list of requirements, install them and copy all the "recipes" needed to configure the infrastructure.
– System Configured: This process implies the installation of all the needed packages to install the EMI software in all the nodes and the whole configuration process. Since the image base selected is very lightweight a large number of software packages must be installed (254 packages and 112 MB). This process is made simultaneously in all the nodes, so it may cause some bottlenecks in the network or in the disk access.

In the node addition test and the following reconfiguration of the infrastructure the steps are the following:

– VM Running: As stated before, using a small image and EBS this time is quite short.
– System Reconfigured: This step includes the time needed for the VM to boot and have the SSH server active, and the configuration of the added node and the reconfiguration of the rest of the nodes.

Table 2 shows the time needed in each individual step and Figure 7 depicts the accumulated time needed to perform all the steps. The shown times are the average values of three tests performed in each case. In the case of EC2 the measures obtained slightly change for the different experiments and only depend on the network performance with the different software repositories used. In the OpenNebula one, as a small Cloud platform the are many factors as the number of VMs launched, the overload on the network or the disks of the physical nodes etc. As shown in the results the average time needed to have a small or medium sized fully functional EMI-2 cluster is about 20 or 30 minutes and the most time consuming step is the configuration since the EMI software has to install a large list of packages in every node. The time needed to add a new node is slightly lower as only one machine must be totally configured and the other nodes just need to add the new one to the configuration. Finally, the node removal is the quickest

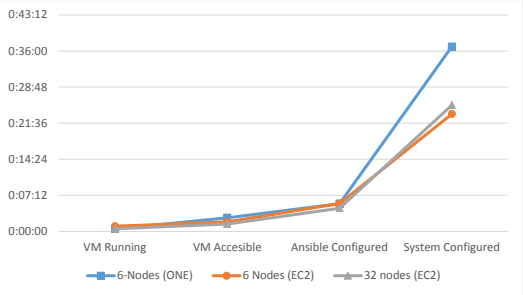

Fig. 7 Virtual Infrastructure Creation Times

operation since terminating a VM can be done very quickly and it only needs to remove the node from the configuration of the other nodes.

In this last use case, the time needed to add a node to the virtual infrastructure is reasonably long. In this case the system uses a basic Scientific Linux image as VMI, so the contextualization process must install a large number of packages. This is not a limitation of the IM since this use case fits many scenarios (dedicated nodes for specific experiments, maintenance interventions, training infrastructures, development environments, software verification, etc.). If deployment time is an issue (e.g. dealing with an unpredicted workload), the configuration process could be reduced by using a pre-configured VMI with some (or most) of the software requirements.

4.2 Higher level services - EC3

In the EC3 work [9] the IM is used in conjunction with a energy management software for clusters called CLUES [2] to enable the deployment of elastic clusters on cloud platforms.

In this case it is used the IM functionality to launch virtual infrastructures on cloud providers and the management of the cluster sizes automatically on demand provided by CLUES.

CLUES and the IM interact at two levels: First in the initial launch of the front-end VM of the elastic cluster. In this step the EC3 launcher starts an IM that becomes responsible of deploying the cluster front-end. This is done by means of the following steps:

1. Selecting the VMI for the front-end. The IM can take a particular user-specified VMI, or it can contact the VMRC to choose the most appropriate VMI available, considering the requirements specified in the RADL.
2. Choosing the cloud provider according to the specification of the user (if there are different providers).
3. Submitting an instance of the corresponding VMI and, once it is available, installing and configuring all the required software that is not already preinstalled in the VM: The Local Resource Management System (LRMS) selected by the user and CLUES configured to use this LRMS and the IM to manage the virtual nodes.

One of the main LRMS configuration steps is to set up the names of the cluster nodes. This is done using a sysadmin-specified name pattern (e.g. vnode-*) so that the LRMS considers a set of nodes such as vnode-1, vnode-2... vnode-n, where n is the maximum cluster size. This procedure results in a fully operational elastic cluster.

In the second level CLUES internally uses an IM to submit the VMs that will be used as working nodes for the cluster. For that, it uses a RADL document defined by the sysadmin, where the features of the working nodes are specified. Once these nodes are available, they are automatically integrated in the cluster as new available nodes for the LRMS. Thus, the process to deploy the working nodes is similar to the one employed to deploy the front-end.

Notice that the ability to dynamically deploy, configure, monitor and re-configure virtual infrastructures at runtime paves the way for a simplified usage of Cloud resources for different computational activities.

5 Conclusion and Future Work

This paper shows a platform for the dynamic management of virtual infrastructures. The platform developed considers all the aspects related to the creation and management of virtual infrastructures: the expression of the user requirements using the RADL language, the deployment of the VMs on the desired platform and all the configuration (and reconfiguration) tasks to make the infrastructure work properly.

One of the main goals of this platform is to enable the users to reuse the existing VMIs created by themselves or by other users, thus avoiding the work of creating new VMIs for each application to be launched in cloud environments. For this purpose it has been used a catalog system to index the VMIs with all the relevant metadata, including the list of software installed. This is combined with a contextualization process, which enables the installation and configuration of all the user requirements. This combination enables to contextualize in run-time all the required applications where the installation and configuration process can be done automatically in a reasonable time, and in the other cases have a prepared VMI correctly registered in the catalog with the application with the complex installation produce make by hand.

Another important feature is the possibility to access different cloud providers. For this reason, a modular system has been developed to interact with both public and on-premise cloud providers. It enables the management of virtual infrastructures in the same way as using any underling platform. Therefore, a user can define an RADL and test it in a local virtualization platform or a private cloud, and then without any modification deploy a replica of the same infrastructure using a public cloud platform.

The elasticity management, as one of the key features of the cloud infrastructures, is also supported by the platform. Not only horizontal elasticity, by adding or removing VMs to an infrastructure, but also the vertical one adapting the VM capacities to the application requirements (by dynamic changing the VM memory size).

Some use cases have been shown to demonstrate the functionality of the platform using two different environments (Amazon EC2 and OpenNebula). The first

case study describes using the IM to launch an EMI-2 cluster with Torque. The second one shows the integration of the IM in another software layer (EC3) to elastically manage the size of a virtual cluster.

The IM relies on network connectivity to access all the components involved in the virtual infrastructure deployment: cloud provider, VMs, software repositories, etc. In case of network failures some of the steps of the infrastructure deployment could fail. If the error is produced in the deployment step, the IM will retry it a number of times (configurable), but if the error is produced in the contextualization step, it will fail. In this case the IM will notify about the error, and the user or higher level software will be able to capture it and use the IM recontextualization function to start again the configuration process. Given that Ansible provides idempotent functionality, only the steps that have failed will be made.

Future works will include the development of new plug-ins in the Cloud Connector layer to access more cloud providers such as Microsoft Azure or Google Compute, and to integrate other aggregation libraries as a gateway to additional Cloud systems. Recently, Amazon has presented a new service called OpsWorks (in beta version at the time of the work of this article) that provides a similar functionality to the IM, which could be analysed once it becomes stable. Another interesting issue is the selection of the "best" cloud provider. In this area further research is required to enable cloud-bursting and to include performance/price ratios or similar criteria.

Acknowledgements The authors would like to thank to thank the financial support received from the Ministerio de Economía y Competitividad for the project CodeCloud (TIN2010-17804).

References

1. de Alfonso, C., Caballer, M., Alvarruiz, F., Molto, G., Hernández, V.: Infrastructure Deployment Over the Cloud. In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science, pp. 517–521. IEEE (2011). DOI 10.1109/CloudCom.2011.77
2. Alvarruiz, F., De Alfonso, C., Caballer, M., Hernández, V.: An Energy Manager for High Performance Computer Clusters. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, pp. 231–238 (2012). DOI 10.1109/ISPA.2012.38
3. Amazon Web Services: AWS CloudFormation (2013). URL `http://aws.amazon.com/es/cloudformation/`
4. Apache: Whirr (2013). URL `http://whirr.apache.org/`
5. Blanquer, I., Brasche, G., Lezzi, D.: Requirements of scientific applications in Cloud offerings. In: Proceedings of the 2012 Sixth Iberian Grid Infrastructure Conference, IBERGRID '12, pp. 173–182 (2012)
6. Bresnahan, J., Freeman, T., LaBissoniere, D., Keahey, K.: Managing appliance launches in infrastructure clouds. In: Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery, TG '11, pp. 12:1—-12:7. ACM, New York, NY, USA (2011). DOI 10.1145/2016741.2016755
7. Buyya, R., Ranjan, R., Calheiros, R.N.: InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. Algorithms and Architectures for Parallel Processing **6081**, 20 (2010)
8. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6), 599–616 (2009). DOI 10.1016/j.future.2008.12.001

9. Caballer, M., De Alfonso, C., Alvarruiz, F., Moltó, G.: EC3: Elastic Cloud Computing Cluster. Journal of Computer and System Sciences (2013). DOI 10.1016/j.jcss.2013.06.005
10. Caballer, M., García, A., Moltó, G., de Alfonso, C.: Towards SLA-driven Management of Cloud Infrastructures to Elastically Execute Scientific Applications. In: 6th Iberian Grid Infrastructure Conference (IberGrid), pp. 207–218 (2012)
11. Carrión, J.V., Moltó, G., De Alfonso, C., Caballer, M., Hernández, V.: A Generic Catalog and Repository Service for Virtual Machine Images. In: 2nd International ICST Conference on Cloud Computing CloudComp 2010 (2010)
12. Cuomo, A., Modica, G., Distefano, S., Puliafito, A., Rak, M., Tomarchio, O., Venticinque, S., Villano, U.: An SLA-based Broker for Cloud Infrastructures. Journal of Grid Computing **11**(1), 1–25 (2012). DOI 10.1007/s10723-012-9241-4
13. DeHaan, M.: Ansible (2013). URL `http://ansible.cc/`
14. Distributed Management Task Force, Inc: Open Virtualization Format (OVF) (2010). URL `http://dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf`
15. Distributed Management Task Force, Inc: Cloud Infrastructure Management Interface (CIMI) Model and REST Interface over HTTP Specification (2012). URL `http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf`
16. EGI.eu: Seeking new horizons: EGI's role for 2020. Tech. rep. (2012). URL `https://documents.egi.eu/public/RetrieveFile?docid=1098\&version=4\&filename=EGI-1098-D230-final.pdf`
17. Elmroth, E., Tordsson, J., Hernández, F.: Self-management challenges for multi-cloud architectures. Towards a Service-Based Internet. Lecture Notes in Computer Science **6994**, 38–49 (2011)
18. HashiCorp: Vagrant (2013). URL `http://www.vagrantup.com/`
19. Jacob, A.: Infrastructure in the cloud era. In: Proceedings of the 2009 International OReilly Conference Velocity (2009)
20. Juve, G., Deelman, E.: Automating Application Deployment in Infrastructure Clouds. In: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, pp. 658–665. IEEE Computer Society, Washington, DC, USA (2011). DOI 10.1109/CloudCom.2011.102
21. Keahey, K., Freeman, T.: Contextualization: Providing One-Click Virtual Clusters. In: Fourth IEEE International Conference on eScience, pp. 301–308 (2008)
22. Keahey, K., Freeman, T.: Architecting a Large-Scale Elastic Environment: Recontextualization and Adaptive Cloud Services for Scientific Computing (2012)
23. Kecskemeti, G., Kertesz, A., Marosi, A., Kacsuk, P.: Interoperable Resource Management for establishing Federated Clouds. In: Achieving Federated and SelfManageable Cloud Infrastructures Theory and Practice, pp. 18–35 (2012). DOI 10.4018/978-1-4666-1631-8.ch002
24. Kertesz, A., Kecskemeti, G., Oriol, M., Kotcauer, P., Acs, S., Rodríguez, M., Mercè, O., Marosi, A.C., Marco, J., Franch, X.: Enhancing Federated Cloud Management with an Integrated Service Monitoring Approach. Journal of Grid Computing **11**(4), 699–720 (2013). DOI 10.1007/s10723-013-9269-0
25. Loutas, N., Kamateri, E., Bosi, F., Tarabanis, K.: Cloud Computing Interoperability: The State of Play. 2011 IEEE Third International Conference on Cloud Computing Technology and Science pp. 752–757 (2011). DOI 10.1109/CloudCom.2011.116
26. Marshall, P., Keahey, K., Freeman, T.: Elastic Site: Using Clouds to Elastically Extend Site Resources. In: Proceedings of the 2010 IEEE/ACM 10th International Conference on Cluster, Cloud and Grid Computing, CCGRID '10, pp. 43–52. IEEE Computer Society, Washington, DC, USA (2010). DOI 10.1109/CCGRID.2010.80
27. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing **30**(5-6), 817–840 (2004)
28. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. NIST Special Publication 800-145 (Final). Tech. rep. (2011). URL `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`
29. Moltó, G., Caballer, M., Romero, E., Alfonso, C.D.: Elastic Memory Management of Virtualized Infrastructures for Applications with Dynamic Memory Requirements. In: Proceedings of the International Conference on Computational Science ICCS 2013, pp. 159–168. Elsevier (2013). DOI 10.1016/j.procs.2013.05.179
30. Morfeo: Claudia (2013). URL `http://claudia.morfeo-project.org/wiki/index.php/Main_Page`

31. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 (2013). URL http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html
32. OCCI working group within the Open Grid Forum: Open Cloud Computing Interface Infrastructure (2011). URL http://ogf.org/documents/GFD.184.pdf
33. Opscode: Chef (2013). URL http://www.opscode.com/chef/
34. Pawluk, P., Simmons, B., Smit, M., Litoiu, M., Mankovski, S.: Introducing STRATOS: A Cloud Broker Service. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp. 891–898 (2012). DOI 10.1109/CLOUD.2012.24
35. Puppet Labs: IT Automation Software for System Administrators (2013). URL http://www.puppetlabs.com/
36. Redl, C., Breskovic, I., Brandic, I., Dustdar, S.: Automatic SLA Matching and Provider Selection in Grid and Cloud Computing Markets. In: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, GRID '12, pp. 85–94. IEEE Computer Society, Washington, DC, USA (2012). DOI 10.1109/Grid.2012.18
37. Rodero-Merino, L., Vaquero, L.M., Gil, V., Galán, F., Fontán, J., Montero, R.S., Llorente, I.M.: From infrastructure delivery to service management in clouds. Future Generation Computer Systems **26**(8), 1226–1240 (2010). DOI 10.1016/j.future.2010.02.013
38. StratusLab: Claudia Platform (2013). URL http://stratuslab.eu/doku.php/claudia
39. Sundareswaran, S., Squicciarini, A., Lin, D.: A Brokerage-Based Approach for Cloud Service Selection. In: Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing, CLOUD '12, pp. 558–565 (2012). DOI 10.1109/CLOUD.2012.119
40. Telefónica Investigación y Desarrollo S.A. Unipersonal.: Telefónicas TCloud API Specification. (2010). URL http://www.tid.es/files/doc/apis/TCloud_API_Spec_v0.9.pdf
41. Yangui, S., Marshall, I.J., Laisne, J.P., Tata, S.: CompatibleOne: The Open Source Cloud Broker. Journal of Grid Computing (2013). DOI 10.1007/s10723-013-9285-0