2011 Third IEEE International Conference on Coud Computing Technology and Science

Infrastructure deployment over the Cloud

Carlos de Alfonso, Miguel Caballer, Fernando Alvarruiz, Germán Moltó and Vicente Hernández
Instituto de Instrumentación para Imagen Molecular (I3M).
Centro mixto CSIC – Universitat Politècnica de València – CIEMAT.
Camino de Vera s/n, 46022 Valencia, España
{caralla, micafer1}@upv.es, {fbermejo, gmolto, vhernand}@dsic.upv.es

***Abstract*—With the advent of cloud technologies the scientists have access to different cloud infrastructures in order to deploy all the virtual machines they need to perform the computations required in their research works. This paper describes a software architecture and a description language to simplify the creation of all the needed resources, and the elastic evolution of the computing infrastructure depending on the application requirements and some QoS features.**

Keywords- cloud; infrastructure manager; Cloud computing

I. INTRODUCTION

This paper describes a software architecture designed in order to make it easy for scientists to create and to use computational infrastructures provisioning resources from different Infrastructure as a Service (IaaS) cloud providers.

To that end, different component and services have been developed, dealing with aspects such as: i) the description of the infrastructure required for the end-user application; ii) the search of the most suitable Virtual Machine Images (VMIs) that are available to build such computing infrastructure; iii) the process of effectively deploying the Virtual Machines (VMs) in local or remote cloud providers; iv) the contextualization of the VMs for the successful execution of the applications.

These different aspects mentioned above dictate the organization of the paper. First, we present the *Resource and Application Description Language* (RADL), a language for the end-users to describe the computational infrastructure needed to run their applications. Section 3 discusses the contextualization process, which automatically installs and configures the required software dependencies to execute the end-user applications. Section 4 presents the Infrastructure Manager, which orchestrates the different components, enabling the effective deployment of an initial computing infrastructure, and the further operations to modify it on demand, adding or removing virtual nodes. A case study for the creation of a Hadoop cluster is shown in section 5. The conclusions and future work are provided in section 6.

References to previous works related to the areas covered by this paper are provided in the corresponding sections.

II. RADL

There are several languages that can be used to describe virtual machines or appliances. One of the most prominent is the Open Virtualization Format (OVF) [1], which provides a very detailed description of the virtual system. OpenNebula [2] also provides a language for describing single VMs.

It is also possible to provide a low level description of a VM hardware using vendor specific languages (e.g. VMWare VMX files) that are easier to deal with. But they are too specific for describing the hardware of a single VM and leave apart aspects such as the applications that are installed in the VM or the network connections between the VMs. Therefore, these languages are not suitable for a high level definition of a set of VMs that should be able to interact with each other. Moreover, the use of any of these languages is unsuitable for the end user, because he/she would have to deal with low-level aspects (e.g. the bus for the disk), and would not be able to use high-level constructs (e.g a 20 node MPI-based cluster managed by PBS/Torque).

The RADL aims at describing at a higher level the VM infrastructure that a user needs for a specific task. The purpose of the RADL is to describe the features that a given virtual infrastructure must offer instead of the virtual hardware. It is based on well-known languages and standards, borrowing their principles and abstractions, and putting them together into a more user-friendly language. In this sense, most of the semantic of the fields that describe the requirements for the VM has been taken from OVF in order to map them into the VM creation document. The principles for the separation of the environment from the VM and its integration have been taken from the OCCI (Open Cloud Computing Interface) [3] Open Grid Forum standard. The language itself is pretty much inspired in the syntax of the well-known Condor's ClassAD language [4].

The result is a declarative language that describes virtual infrastructures, by declaring the features or requisites of the VMs to be deployed. Using these requisites, a description of the VM (using OVF or a vendor specific language) must be created to finally deploy the VM.

RADL acts as a query language that must be processed by an infrastructure deployment component to deal with the features offered by the Cloud deployments to instantiate the actual network configuration, to build the VMs, and to configure them to fulfill the requirements of the user.

The RADL provides a very comprehensive mechanism for the user to describe his infrastructure by defining the virtual hardware requirements of the application, but also the software that must be available on each of the VMs. In this sense, we have defined three types of constraints for the infrastructure: *environmental features*, *virtual hardware features* and *software features*. A RADL document must be

978-0-7695-4622-3/11 $26.00 © 2011 IEEE
DOI 10.1109/CloudCom.2011.77

understood as a SLA that is likely to be accomplished by different components in the infrastructure. The stated features may be met by the available VMIs, or be provided by means of contextualization procedures [5], but also the Cloud platform in which the VMs are deployed must be able to provide the type of resources requested by the user. Additionally, the user will probably have some constraints on the maximum budget that he can afford for the infrastructure, the VMIs that he is allowed to use, etc.

A. RADL document structure

A RADL document consists of the declaration of the different elements that will compose the infrastructure and the desired features. The document includes environmental features and the different types of VMs.

1) Environmental features

The environmental features consist of devices, services, etc. that are not provided by the VMs. The existence of such elements should not depend on the deployment of the VMs. The VMs must be able to link to such environment. Some examples are connectivity networks, windows domain services or Storage Area Networks.

Currently we are only considering the *network* environmental feature: a Local Area Network (LAN) to which the VMs may be linked. The "network" keyword is used, followed by the next features:

- Outbound: indicates whether the VMs linked to the network have access to the public network or not.
- Address: the IP addresses and network mask the VMs will receive from a DHCP server.

2) Virtual Machine Types and Software Features

The virtual infrastructure to be deployed is described by defining the type of VMs and how many of each type must be actually deployed. In this case, the *system* keyword is used to define a template with the set of the features that a VM must meet. Each template has a *name*, a set of *features* and a *number* of instances to be deployed.

```
system <name> [(<features>)] <number>
```

The features that a virtual system must meet are stated by an expression with the following grammar.

```
<expression> := <expression> <bool_connector>
<expression> | <attribute> <operator> <value> |
soft <punctuation> ( <expression> ) | attribute
<contains> ( <tuple> )
<tuple> := field <operator> <value> | <tuple>
<bool_connector> <tuple>
<punctuation> := integer
<bool_connector> := and | or
<operator> := = | > | < | >= | <=
<value> := ' string ' | number [ <qualifier> ]
<qualifier> := M | G | K
```

At the end, such expression is a set of requirements for the virtual hardware components and some extra information about the VM contents. These components and metadata are referred to as *attributes* and they are expressed as LDAP entries that gather the class hierarchy for each attribute.

Some of the attributes of a VM are multivalued. In some cases the order of such elements is important since it may be a decision factor on how the system behaves (e.g. the boot disk). In such cases the RADL follows the LDAP syntax for numbering the entries of an array-like attribute (e.g. attribute.0.field, attribute.1.field, etc.). In cases that the components contained by an attribute are not orderable we use the "*contains*" keyword to reflect the collection-like behavior (e.g. the applications that are installed in a VM).

The user should be aware that it could be difficult to create a VM that meets all of the specified requirements. The "*soft*" expression enables to express that a requirement is interesting to be fulfilled but it is not mandatory. This expression is accompanied by a "*rank*" that is an integer that will be computed to get a final score for the VMI, expressed in a user-defined scale. The total score must be interpreted as "the goodness" of the VMI for the user, and it will help to select the most suitable VMI to deploy the VM. The requirements that are not under the influence of the "*soft*" construction are considered to be indispensable and if they are not met, the VM must not be deployed.

We are currently considering several LDAP *ObjectClasses* that represent the components of a virtual machine that are represented by the next keywords:

- system: It refers to the metadata associated to the VM (e.g. the virtualization subsystem: KVM or Xen).
- cpu: It represents the features of the virtual CPUs, with the attributes "*count*" and "*arch*" (e.g. i386)
- memory: It refers to the RAM memory of the VM. Its attribute "*size*" represents the amount of memory.
- net_interface: It is an array of network interfaces (referred as net_interface.0, net_interface.1, etc.). Each interface contains an attribute named "*connection*" that links to a LAN by using its name. Other attributes considered are "*ip*" and "*mac*".
- disk: It is an array of disks (i.e. disk.0, disk.1, etc.) where the disk zero will act as the boot disk.

Each of the attributes included above are related to the *virtual hardware features* of the VM and they should be negotiated with the cloud deployment to decide the actual values that should be used. The *software features* are related to the content of the disks that are attached to the VM that are, in fact, mapped to VMIs. The disks have some attributes that refer to the content of such images:

- *free_size:* amount of free space in the disk.
- *os*: operating system the VM will boot in case that the VMI is attached as the boot disk. It has four fields: (1) *name* (e.g. linux, windows, etc.); (2) *flavor* (e.g. ubuntu, fedora, SP3, etc.); (3) *version,* (e.g. XP, 7, 9.10, etc.); and (4) *credentials* that enables to configure the access method to the VM by specifying a *user* and a *password*, or a *public key* for ssh access.
- *applications*: the collection of applications installed in the disk. Each of the elements has different fields: *name* (unique name for the application), *version, path* (where the application is installed), etc.

Most of the attributes included in the document are designed to be directly mapped to OVF attributes. Therefore, their values must be set according to the semantic defined by such standard. Other attributes (e.g. the name of the applications or the style for the version numbering) will be specific for

the cloud deployment and the Infrastructure Manager that will interpret the RADL. It is advisable to follow the OVF recommendations for the name of the applications, but also for other items that are not officially standardized.

III. CONTEXTUALIZER

Once decided the most appropriate VMI it is important to automatically deploy the remainder software dependencies to execute a given scientific application. For example, a Matlab application requires the Matlab Runtime just like a Java application requires the Java Runtime.

The deployment of software in VMs is usually performed manually. In fact, the deployment of an application in a master VMI will exist in the different VM instances. However, the coordinated usage of different providers that can support a myriad of diverse hypervisors, especially in the context of sky computing [6], demands automated software deployment. In these scenarios, VMIs cannot be easily transformed to support multiple hypervisors. Therefore, automated software deployment is a relevant task that eases the adaptation of the scientific application to the Cloud.

There exist tools to perform the automated deployment of applications, such as Puppet, Chef or Capistrano, but they do have dependencies on other software packages, thus requiring a complex deployment. Moreover they are conceived for large infrastructures where the state must be maintained over the time while in cloud environments it is commonly required to simply achieve the specified configuration stated by the user.

Instead, we have developed a lightweight and portable contextualizer [7] that only depends on the standard Python language, which is available in virtually every GNU/Linux distribution. This tool enables the unattended execution of commands specified in an XML document in order to perform the automated installation of software dependences and the application itself. The user can create an Application Deployment Description (ADD) to specify how to deploy the application (the compilation approach, the configuration, some required post-processing, etc.). There are ADDs already created for common software. Therefore, this tool performs the installation of the software so that the VM is configured to be able to successfully execute the application.

The contextualizer can work as standalone software or in a client-server fashion where ADDs and software packages are dynamically retrieved on-demand from its server counterpart in order to reduce the size of the contextualization package for a VMI to the bare minimum. This mechanism is suitable for specific deployments such as computing clusters where only one of the nodes is accessible by a public IP address.

IV. INFRASTRUCTURE MANAGER

The main goal of the Infrastructure Manager (IM) is to provide a set of functions to enable the effective deployment of an initial computing infrastructure, and the further operations to modify it on demand, adding or removing nodes. The IM offers an API with a reduced and simple set of functions enabling the creation of the infrastructure, getting the information about the VMs and the addition or removal of VMs in the deployment (RADL is used to specify the requirements of the VMs).

A. Architecture

Fig. 1 depicts the architecture of the IM. The upper level shows the applications that use the functions provided by the API to connect to the manager.

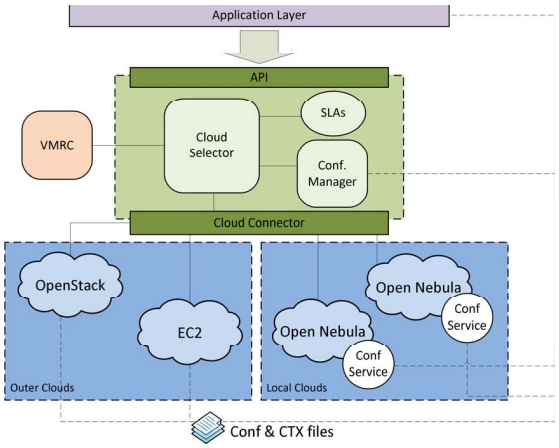

Figure 1. Infrastructure Manager Architecture

The Cloud Connector (CC) layer provides a set of functions enabling to homogeneously access different cloud middlewares (OpenNebula, OpenStack, EC2, etc.). These functions provide the information needed by the IM about the cloud infrastructure VMs: available cores or memory, VM image types supported, etc. The drivers also abstract the deployment of VMs in each IaaS provider. A CC driver has been developed for OpenNebula, and work is in progress to develop an EC2 driver.

The core component of the IM is the Cloud Selector (CS). It is in charge of selecting where to deploy the VMs specified in the RADL document. This central component will contact the rest of services of the architecture, orchestrating the deployment of the infrastructure.

In order find a suitable VMI that accomplishes the requirements of the user and is compatible with the available cloud system, the Virtual Machine Catalog and Repository system (VMCR) [8] has been used. This component stores and indexes VMIs in order to be reused in multiple contexts. It also implements matchmaking algorithms to obtain the VMIs that satisfy a set of requirements.

The CS contacts the VMCR to choose the best available VMIs, considering the requirements specified in the RADL document. The CS also uses the CC layer to obtain all the information about the available cloud infrastructures. At this point the CS selection task is two-fold. It must select the best image from the results provided by the VMCR and the best cloud infrastructure. In some cases the best images could not be compatible with the best cloud deployment, and the CS must balance in order to select the best combination to achieve the best results.

The CS must also take into account the different SLAs of all the components of the systems. The SLA is a broad concept that must cover many kinds of system features:

- Authorization: the CS must check if a user has proper credentials to access the different clouds.
- Cloud features: The list of capacities provided by the cloud infrastructure to the user: number of cores, available memory, transfer bandwidth, etc.
- Economics: The prices of the different infrastructure components. If different cloud infrastructures are available, the economics are a key issue in order to select the most suitable resources at the lowest price.
- Co-allocation features: Some applications may require all the VMs to be deployed in the same cloud infrastructure. Other applications without these restrictions can use mixed cloud infrastructures.

Other important aspects related to the quality of service (QoS) must be considered when selecting the cloud (or clouds) where the deployment is made (e.g. selecting the closest cloud infrastructure and the VMI to minimize the time to transfer the image, or selecting the VMI with the specific format needed by the hypervisor to avoid an additional step to convert the image).

Finally the last component is the Contextualizer and Configuration System (CCS). This element is in charge of the installation and configuration of all the required software in all the VMs in order to accomplish the requirements of the RADL document. The CCS has two different components:

The first one is the Configuration Manager (CM) that coordinates the installation among different cloud deployments. It stores historical data about the installation of the software in order to provide information to the IM about the time needed to install some specific software.

The second component is the configuration service that is in charge of the effective installation of the software in the VMs. This service provides a small set of functions to ease the creation of the VAs by installing and configuring all the required software in the VMs. It can also configure the set of VMs as a whole to work as a cluster. These services can also be used from the application layer in order to install or configure some application specific features in the VMs.

The configurator service provides three functions: the first one enables *ssh* access without password among a set of machines (usually required in the configuration of any cluster). The second function is an interface to launch the contextualizer in the specified nodes, enabling to install and to configure the required software. The third function provides a list of the installable applications to the IM.

This service uses two approaches to contextualize and configure the VMs. The first one is used in the local clouds, where the configuration service is deployed as part of the architecture. In this case the application must contact this service to configure all the VMs. The second approach is used in the case of external clouds where the configuration service cannot be deployed due to the provider's restrictions. This approach is based on the idea that every deployed cloud infrastructure will have at least one VM with a public network interface. In these cases the IM connects to this service by the public interface and contextualizes and configures all the VMs.

B. *Functionality*

The steps needed to deploy an infrastructure in a cloud environment using the IM are the following:

1. The application layer provides the IM with the RADL describing the features of the infrastructure.
2. The IM uses the CS to contact the VMCR in order to select the most suitable images.
3. The CS contacts the CC to get the information about the available cloud deployments.
4. The CS contacts the CM to get information about the available applications that can be installed in the VMIs. The IM combines the information about the VMIs, the cloud list, and installable applications with the SLAs information, trying to get the best combination of image – cloud deployment to create the infrastructure denoted by the RADL document (the current version of the IM only considers the QoS parameters related to the proximity of the VM image to the cloud infrastructure). If the IM detects that any of the applications is not installed in the VMIs, and cannot be installed, it will notify the caller application.
5. In some cases the CS may select an image that doesn't have all the applications requested in the RADL document, and the IM must use the CCM in order to install all the needed applications. The CCM must test if the configurator service is enabled in the cloud deployment selected. If it is, the CCM will contact it. Otherwise the CCM must obtain a public IP address of one of the launched VMs in order to connect to it to launch the command-line version of the configurator.
6. At this point the IM has deployed the infrastructure. The caller application can contact the IM to obtain all the information required to access and use the infrastructure.
7. The last step is application dependent, so the upper level performs it. The application must contact again the configurator service in order to configure the software installed in the nodes, e.g. to configure a PBS cluster, or a Hadoop cluster, or some other specific software.

V. CASE STUDY: CREATE A HADOOP CLUSTER

To demonstrate the functionality of the IM, it has been used to automatically deploy a Hadoop cluster consisting of 5 nodes. A high level application has been developed that uses the functionality provided by the IM. The application uses the next RADL document to create the infrastructure:

```
network publicNet ( outbound = 'yes')
network privateNet
system ( cpu.arch='i686' and cpu.count>=1 and
memory.size>=512M and
net_interface.0.connection='publicNet' and
net_interface.1.connection='privateNet' and
disk.0.os.name='linux' and
disk.0.applications contains (name='hadoop')
) 1
system ( cpu.arch='i686' and cpu.count>=1 and
memory.size>=512M and
net_interface.0.connection='privateNet' and
disk.0.os.name='linux' and
disk.0.applications contains (name='hadoop')
) 4
```

This RADL file describes two different kinds of machines. Both of them have similar requirements (at least one 32-bit CPU, 512 MB of RAM, with Linux OS and Hadoop installed). The only difference between them is that the first system has two network interfaces (a public one and a private one), while the second system has only a private network interface.

We used a testbed with two different cloud deployments using OpenNebula 2.2 software. The first one (*kefren*) has 18 Xeon dual-processor nodes and uses VMware hypervisor. The second cluster (*dellblade*) is composed of 4 dual-processor nodes with 4 cores and the KVM hypervisor.

The VMCR service has two registered Linux images. The first one is a VMware image located in the *kefren* front-end node and the second one a KVM image located in the *dellblade*. Both images are a fresh Ubuntu installation.

These are the steps needed to complete the process of deploying the fully functional Hadoop cluster using the IM:

1. The client application sends a "create infrastructure" request with the previous RADL document.
2. The IM contacts the VMCR and obtains the list of the available images. None of them has Hadoop installed.
3. The CS contacts the CC to obtain the information about the cloud infrastructures available.
4. The CS selects dellblade and the image located in the dellblade front-end node, because it is the most powerful cloud infrastructure.
5. The IM contacts the CC to submit the VMs.
6. The IM must contact the configurator and contextualizer service in order to install Hadoop. Java is also installed because it is a requisite in the Hadoop installation. All the installation is managed by the contextualizer.
7. Once the VMs are fully installed, the application must contact the configurator in order to configure all the individual VMs as a Hadoop cluster.
8. We have a fully functional Hadoop cluster! At this point the caller function can connect to the front-end node to operate with the Hadoop cluster.

From the point of view of the client application that contacts the IM, it must only call the function to create the infrastructure, and periodically call the function to get the information about the infrastructure. This function informs about the status of the deployment process and gives all the information needed to connect with the VMs once they have been correctly deployed.

VI. Conclusions and Future Works

This paper described a software architecture designed to simplify the management of infrastructures in Cloud environments, enabling the scientists an easy access to Cloud technology. Different elements composing the architecture have been addressed: The RADL language for describing the features required by the virtual computing infrastructure. The VMCR manages information about VMIs to be used in the deployment, enabling to select the most suitable image to create the virtual appliances. If these images lack some required software, the contextualizer component enables to easily install the required software, with minimal user intervention. Finally, the IM has been described as the core component of the architecture. This central element connects to the rest of the components in order to orchestrate the creation of the infrastructure.

This work describes the initial version of the IM. As a prototype it must evolve to achieve all the objectives of the proposed architecture. It currently considers only some simple SLA elements such as the user authorization. Other CC connectors will be added to expand the functionality possibly by means of aggregation APIs such as jClouds or Apache LibCloud. Moreover automatic elasticity features can be introduced, to scale out or scale in the infrastructure according to the resources needed at each time. Also creating algorithms for the selection of the proper cloud deployment is an issue to be addressed.

The usage of abstraction mechanisms for automated deployment of self-configurable infrastructures for the Cloud paves the way for an enhanced interoperability. This is of special importance in sky computing scenarios, with multiple Cloud providers. Therefore, the development of generic and customizable components for the Cloud contributes to the ecosystem of tools to embrace the adoption of Cloud technologies by the scientific community.

Acknowledgment

The authors wish to thank the financial support received from The Spanish Ministry of Science and Innovation to develop the project "Servicios avanzados para el despliegue y contextualizacion de aplicaciones virtualizadas para dar soporte a modelos de programación en entornos cloud", with reference TIN2010-17804, and to the "Vicerrectorado de Investigación de la Universitat Politècnica de Valencia" for the project PAID-06-09-2810.

References

[1] Crosby, S., et al.: Open Virtualization Format Specification (OVF). Technical Report DSP0243, Distributed Management Task Force, Inc. (2009)

[2] Fontán, J., Vázquez, T., Gonzalez, L., Montero, R. S., and Llorente, I. M.. "OpenNEbula: The open source virtual machine manager for cluster computing", in *Open Source Grid and Cluster Software Conference* - Book of Abstracts, San Francisco, USA. (2008).

[3] OCCI working group within the Open Grid Forum. "Open Cloud Computing Interface – Infrastructure" (http://ogf.org/documents/GFD.184.pdf)

[4] Raman, R., Livny, M., Solomon, M. "Matchmaking: Distributed Resource Management for High Throughput Computing." in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pp. 28-31, (1998).

[5] Keahey, K. and Freeman, T., "Contextualization: Providing One-Click Virtual Clusters," in *Fourth IEEE International Conference on eScience*, pp. 301-308. (2008)

[6] Keahey, K., Tsugawa, M., Matsunaga, A., and Fortes, J., "Sky Computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43-51. (2009).

[7] Moltó, G., Hernández, V.: "Management and Contextualization of Scientific Virtual Appliances", in *Cloud Futures 2010: Advancing Research with Cloud Computing*. (2010)

[8] 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).