Standard Interfaces for Open Source Infrastructure as a Service Platforms

Andrei IONESCU
Bucharest University of Economic Studies
andrei.ionescu@antiferno.ro

To reduce vendor lock-in and fragmentation and to evolve into a transparent platform, IaaS platforms must adhere to open standards. All the major open source IaaS Platforms offer interfaces compatible with the de facto standards but mostly lacks support for the de jure, open efforts of standardization. Available implementations of open standards are not part of their main development efforts. More development resources as well as consolidation of open standards are needed to achieve increased portability and interoperability.
***Keywords**: Cloud Computing, RESTful API, interfaces, Open Cloud Computing Interface, Cloud Data Management Interface, Cloud Infrastructure Management Interface, Open Virtualization Format*

1 Introduction

Cloud Computing is an already established technology, already having lost its aura as one of the hottest and most rapid developing topics in the industry. At the base of its services stack, Infrastructure as a Service (IaaS) model utilizes well understood architectures, providing access, mostly, to the same type of computing and storage resources across all platform providers. New topics entered the spotlight, such as integrated management of IaaS platforms, selection of an IaaS platform, vendor lock-in, interoperability and identity in the Cloud and efforts were made to address them by defining and using standardized interfaces. Standard data models and technologies compatible with both the IaaS platforms and the existing Internet Infrastructure had to be used. This led to adoption of established standards and technologies such as XML web services or JSON over RESTful services.

2 Open Cloud Computing Interface

Open Cloud Computing Interface (OCCI) is a collection of community generated open specifications built through Open Grid Forum [1]. Intended to be an open and interoperable RESTful protocol and an API for all cloud-related management activities, it started with a focus on the Infrastructure-as-a-Service layer but later extended to include all the other layers in the Cloud stack.

The specifications are broken into several modules in order to achieve a greater flexibility and extensibility. Separate modules describe

- the core models, defines an abstract representation of real-world resources intended to be manipulated through OCCI renderings [1].
- the rendering of the code model using HTTP/REST, describes the interactions available for an OCCI implementation with the resources built using the core models [2].
- the extensions to the code models specific to implementation of an Infrastructure as a Service API [3], defining its parameters for compute, storage and network.

The main reasons behind the development of OCCI were identified in [4] and [5] as:

- Interoperability, demanding a standardized API and protocol.
- Integration, allowing different service providers to bring together and interconnect platforms based on different technologies.
- Portability, providing standardized data formats understood by different providers, allowing porting between them.
- Innovation, considering that established standards can be a driver for innovation.
- Reusability, Figure 1, working on two levels, first allowing reuse of code through basic standardized APIs and, second, promoting reuse of standards in different

DOI: 10.12948/issn14531305/19.4.2015.06

technology fields.

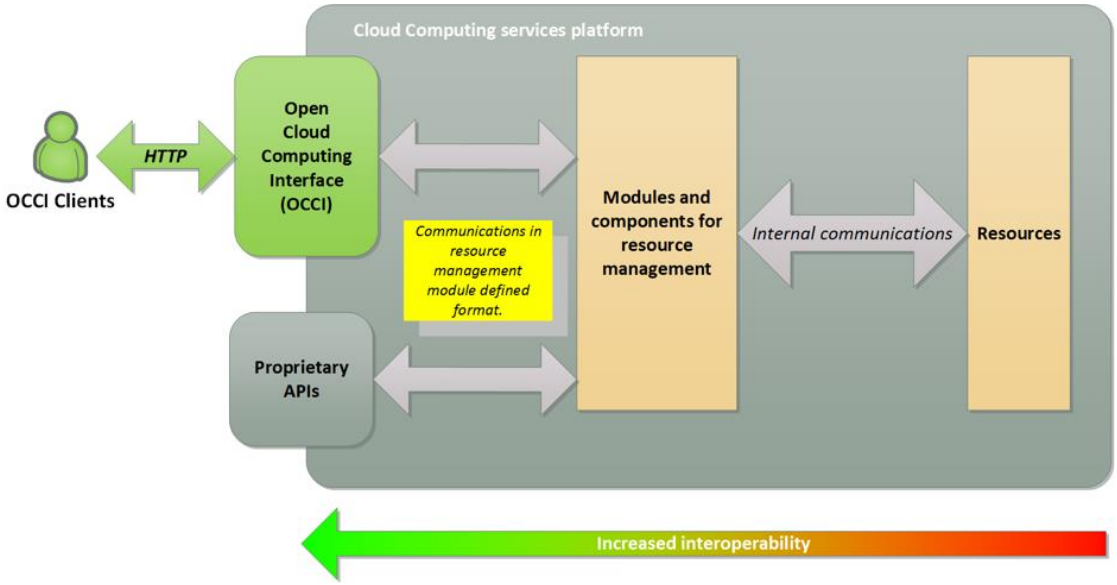

Fig. 1. Open Cloud Computing Interface

3 Cloud Data Management Interface

Cloud Data Management Interface (CDMI) describes a functional interface allowing applications to create, retrieve, update and delete data elements from the Cloud [6]. The standard is developed by SNIA, a global organization of storage solution providers. Using a CDMI compatible interface, cloud data consumers are able to discover the storage features offered by IaaS platforms. Along with data elements, the standard also allows the management of containers, accounts and retrieval of monitoring and billing information, Figure 2.

CDMI is not designed to replace other object access protocols but to complement them. The standard uses RESTful protocol for building its interfaces, to keep it as simple as possible and to encourage its adoption. Adding discovery functions, it allows future extensions to the standard without breaking client compatibility.

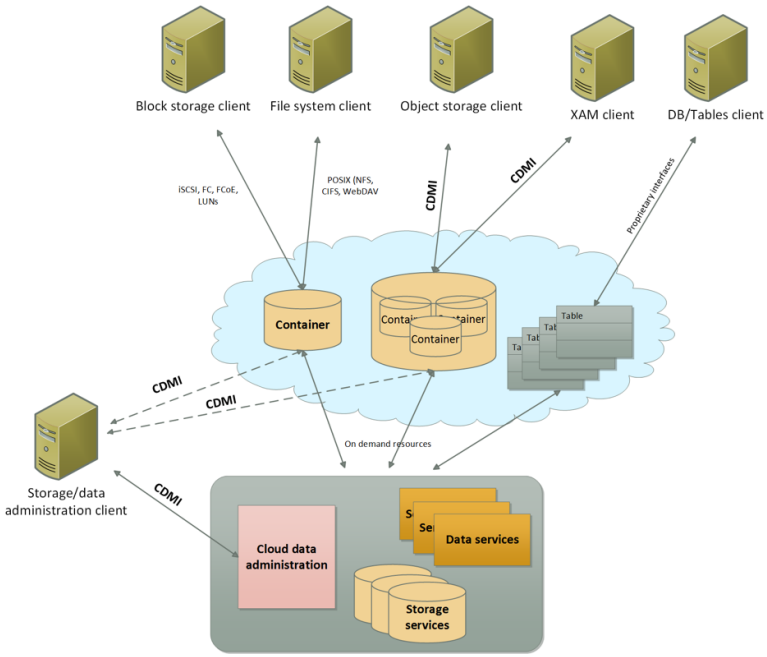

Fig. 2. Cloud Data Management Interface

DOI: 10.12948/issn14531305/19.4.2015.06

CDMI serves as both a storage protocol while, at the same time, offering a layer of client to cloud management and cloud to cloud standardized interactions. Clients can manage credentials to domains defined in the cloud forming a hierarchical structure that have objects attached to them and building a path for accessing and controlling these objects. For cloud to cloud interaction it introduces globally unique identifiers linked to objects for the whole of their lifetime to persist their identity if moved or replicated between clouds. Serialization and deserialization into and from JSON format can be used to transfer objects and their metadata. Primitives are defined which permit the clients to build transfer request indicating source and destination cloud for objects, along with the credentials required for accessing them.

4 Cloud Infrastructure Management Interface

Cloud Infrastructure Management Interface (CIMI) is an open standard providing an API for administering Cloud Computing infrastructures. The standard is maintained by Distributed Management Task Force (DMTF) Cloud Management Working Group, a non profit organization of industry members.

CIMI describes the model and the protocol used by Infrastructure as a Service consumers to interact with the cloud [7], addressing the runtime maintenance and provisioning of cloud services. It uses both JavaScript Object Notation (JSON) and eXtensible Markup Language (XML) to encode communication. The model described by the standard can be mapped to any existing cloud infrastructure and it provides the means for the clients to discover which feature are provided by the cloud implementations.

There are not many CIMI implementations but the Apache Deltacloud is one of them and having drivers for almost all the major Infrastructure as a Service platforms means that any CIMI compatible client is able to interact with most of the deployed clouds.

5 Open Virtualization Format

Another standard maintained by DMTF, Open Virtualization Format (OVF) describe the means to package and distribute software appliances to be run in virtual machines in a hypervisor independent way. Packages distributed using OVF consist of one XML descriptor containing the meta-data which describes the appliance along with its disk images, certificates and auxiliary files.

In the lifecycle of a virtual appliance, Figure 3, OVF covers the packaging, distribution and deployment phases.

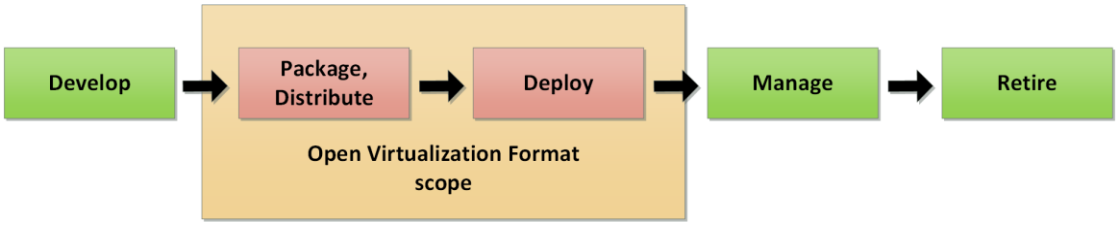

Fig. 3. Open Virtualization Format scope, adapter after [8]

6 Apache Deltacloud

Started in 2010 by Red Hat as a solution for an increasingly heterogeneously cloud interfaces environment, the project intended to be a vendor neutral openly developed API. To escape worries that it will only ever be a single vendor effort, it was proposed as an Apache Software Foundation project and graduated to top level status in 2012.

Deltacloud provides a cloud abstraction, standard REST API. It is not another library but a web service covering functions for management of compute and storage resources. It exposes three different APIs, wrapping their functionalities for 15 Infrastructure as a Service platforms and 7 storage engines [9]:

- Deltacloud classic API.

DOI: 10.12948/issn14531305/19.4.2015.06

- DMTF Cloud Infrastructure Management Interface (CIMI) API.
- Amazon Web Services Elastic Cloud Computing (EC2) compatible API.

7 OpenStack

OpenStack is an open source operating system for the cloud, a collection of projects used to setup and run compute and storage services. Initially developed by Rackspace Hosting and NASA, OpenStack Consortium, the maintainer of the platform with the same name has more than 150 members, including AT&T, Canonical, HP, IBM, Intel and Rackspace.

There are several service families under OpenStack, each with its own API for interfacing with the cloud clients and with the other services, Figure 4.

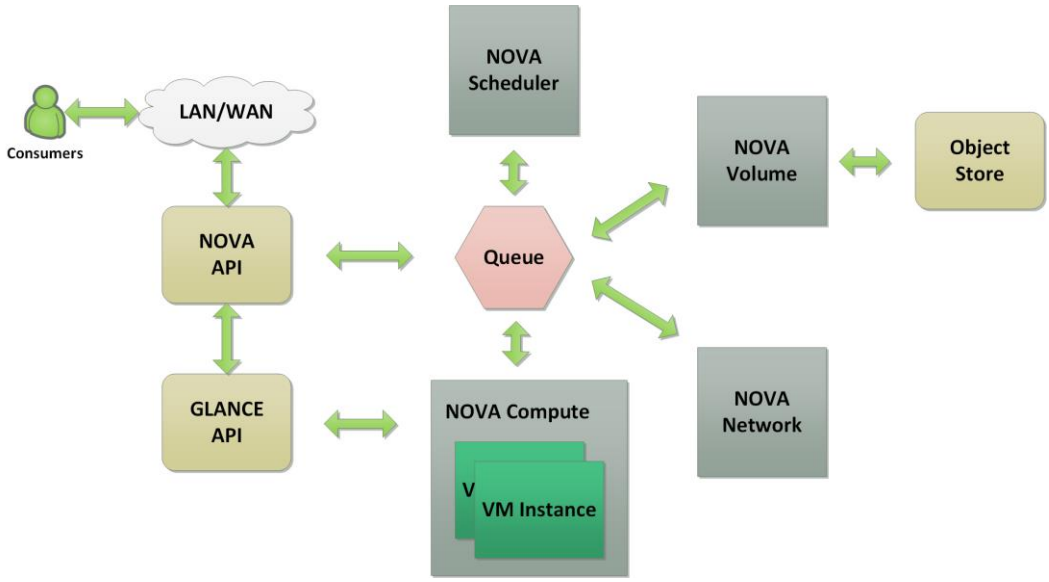

Fig. 4. Basic OpenStack architecture, adapted after [10]

Nova manages the complete lifecycle of virtual machine instances in an OpenStack deployment. It manages the compute and network resources along with the required authorizations for starting, scheduling and stopping virtual machines. Its exposes all its functions through its own web services API as well as a layer compatible with Amazon Web Services EC2. Its API server is the only OpenStack component interacting directly with the outside world.

Glance is the service responsible with storage and retrieval of machine disk images. It can use local file systems, OpenStack's own Object Store services or any storage exposing its functions through an AWS S3 compatible interface.

Swift provides object store services for OpenStack, storing and retrieves data using a RESTful API. It is one of the most mature modules, providing the base services on which Rackspace's Cloud Files services is built [11]. The Swift API is compatible with AWS S3.

Keystone provides authentication and authorization services for OpenStack, managing domains, users and roles. It's a crucial system used by all the other modules using its own REST API (Identiy API).

Horizon provides a web portal for interacting and administering OpenStack. It uses APIs provided by the other services to build the cloud's administrative interface.

Standards

Amazon Web Services EC2 and S3 APIs are natively supported by Nova and Swift modules. Third party Python components provides support for OCCI and CDMI while having an Apache Deltacloud driver offers support for CIMI also. There is no native support for OVF in OpenStack, in order to be executed, virtual machines image files must be manually extracted from OVF packages.

DOI: 10.12948/issn14531305/19.4.2015.06

8 Apache CloudStack

Apache CloudStack is an open-source Infrastructure as a Service platform used to build public and private clouds. It is designed to allow the deployment and administration of big networks of virtual machines while providing the highest availability and scalability. One of its main advantages resides in the fact that it is hypervisor agnostic. It is compatible with Bare Metal, Vmware, KVM, XenServer, Xen Cloud Platform (XCP), vSphere, LXC and Hyper-V [12].

There are multiple ways of administering an Apache CloudStack deployment: through a Web interface, using a full set of command line utilities or using an RESTful API. The platform also provides Amazon Web Services Elastic Cloud Computing (EC2) and Simple Storage Server (S3) compatible API implementations, one of the main reasons behind its development. This allows an easy porting of cloud applications to Apache CloudStack as well as hybrid and federated cloud deployments.

Architecture

Apache CloudStack features a hierarchical architecture, which allows centralized management of a big number of servers through a unique interface, Figure 5. Integration with public clouds implementing Amazon Web Services interfaces is also possible.

In its simplest form, the Apache CloudStack architecture consists of a single central server, optionally having a CloudDB instance running on it. The server's function is to manage the virtual machine instances by collaborating with the hypervisors installed on the Cloud's nodes. These nodes (physical machines) may be located or not in the same data-center and, for ease of administration, are grouped in several levels: regions, zones, pods and clusters.

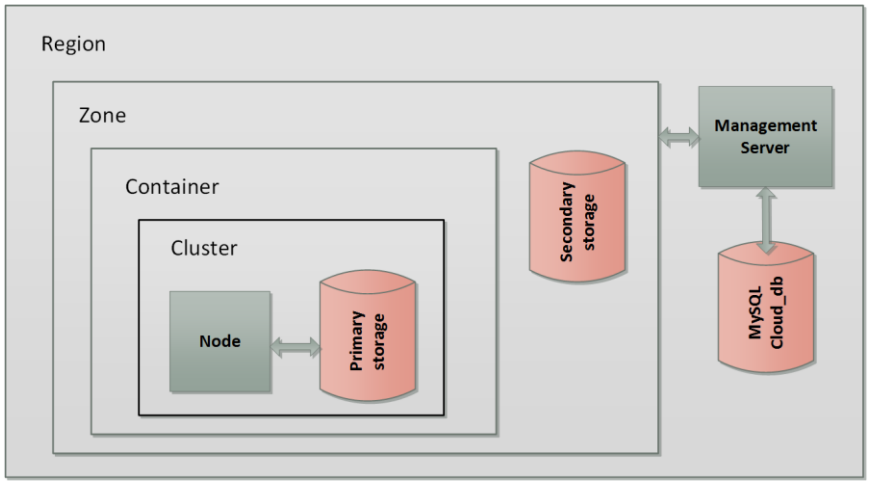

Fig. 5. Apache CloudStack architecture

Regions are linked to the geographical distribution of the physical servers within the cloud. A Region is the highest organizational unit available in an Apache CloudStack deployment. Its administration is made by a server cluster physically located within the region. Virtual machines can be deployed by users in different zones providing a certain level of geographical redundancy. Cloud reactivity is also increased by having the administration server physically closed to the cluster stations as well as deploying virtual machine instances as close as possible to the final end users.

Zones corresponds, usually, to data-centers while still being possible to define multiple zones within the same data-center where they can have separate energy supplies and data lanes, providing a certain level of physical redundancy.

Zones are the highest organizational units available to the users. Starting a virtual machine instance involves selecting a target zone explicitly or having a default one

DOI: 10.12948/issn14531305/19.4.2015.06

selected by default. Images and configurations are not shared between different zones.

A zone has one or more *pods* which consists of one ore more server clusters and at least one storage servers. Pods can be assimilated with racks within the data-center and all its servers are running in the same subnet. Pods are hidden from the end users as the main reason behind grouping servers into pods is the ease of administration.

Clusters are built from servers sharing single hardware specifications and using the same hypervisor. They are using the same subnet, as part of a pod, and use a shared storage space available at pod level. Virtual machine instances can be migrated between servers within the same cluster at runtime.

Hosts are individual physical machines sharing their CPUs, memory, storage and network access for running virtual machine instances. Virtual machine hypervisors are installed on each host. Just as the clusters and pods, they are hidden to the end user. There is no possibility to select a specific host on which a virtual machine instance is to be started.

Storage space

Storage space is segregated between primary and secondary storage. Primary storage is a critical resource, used for executing virtual machine instances along with the application running on them, and its format is dependent on the used hypervisor. Both local (part of VM) and external storage (seen as external volumes mounted by VMs) are part of the primary storage. Secondary storage is used for storing virtual machines' images and snapshots, ISO images, etc.

Resource management

An Apache CloudStack deployment uses a *management server* to administer its resources. Its base functions, as described in [13] are:

- Web user interface available for both cloud administrators and end users.
- CloudStack APIs, both native and compatible with AWS EC2 and S3, using both JSON and XML for data transfer.
- Dispatch virtual machine instances to physical hosts.
- Manage public and private IP account addresses.
- Manage storage space during virtual machines start-up procedure.
- Manage snapshots, disk and ISO images.
- Single point of access for cloud configuration.

DOI: 10.12948/issn14531305/19.4.2015.06

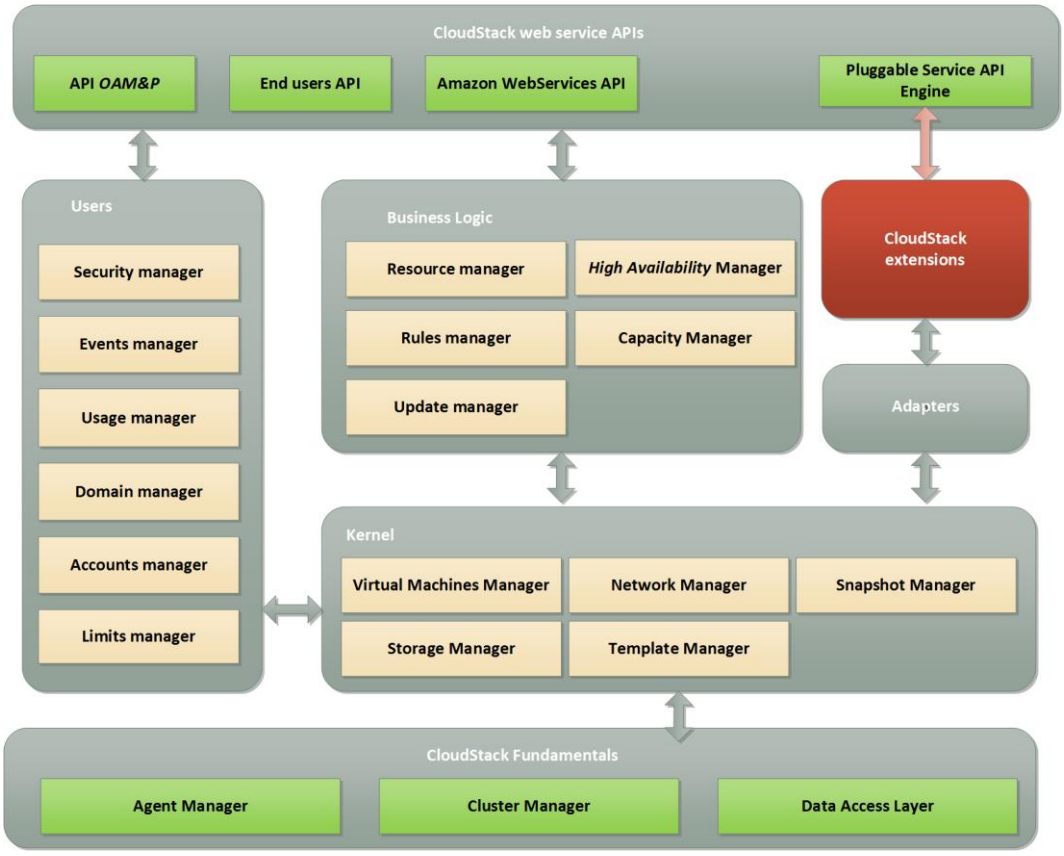

Fig. 6. Apache CloudStack component interfacing, adapted after [14]

The CloudStack APIs, Figure 6 uses three roles [15]:

- Root admin: has access to all the cloud functions, including management of physical and virtual resources.
- Domain admin: has access to all the virtual resources available for the administered domain.
- User: access only to its own virtual machines, attached storage space and network configurations.

An optional component of the resource management stack is the *usage server*, providing aggregate billing data. It also allows the defining of limits and quotas at domain level for the available number of virtual CPUs, RAM, primary and secondary storage space.

Standards

Apache CloudStack implements an estimated 60% of the de facto Amazon EC2 API standard [16]. It has an OCCI implementation through the rOCCI project but no new developments were submitted from the end of 2013. There is no CIMI implementation or translation layer for CloudStack. Open Virtualization Format is also not supported as CloudStack uses native disk images. There is no Apache Deltacloud driver available for Apache CloudStack.

10 Eucalyptus

Eucalyptus is a Linux based software architecture for implementing private and hybrid clouds using in place enterprise architecture [17]. Its name is an acronym for Elastic Utility Computing Architecture for Linking Your Programs to Useful Systems. The main objectives behind its development were [18]:

- Designed from the start for a lean and not intrusive install. It can coexist with other software applications on the same physical machine.
- It has a modular structure. Communication between its modules is language independent and based on open standards. It encourages building of communities centered on the platform.

DOI: 10.12948/issn14531305/19.4.2015.06

- Amazon Web Services EC2 and S3 API compatibility for external interfaces.
- Network virtualization which allows isolation of user generated traffic while also having multiple clusters on the same local network.

Architecture

Eucalyptus was designed to allow interactions using the same tools that were used for Amazon EC2. Each component is implemented as a stand-alone web service exposing its functionalities using WSDL. Each Eucalyptus installation deploys five major components (**Fig. 7**):

- Cloud controller, administrator's interface with the cloud
- Cluster controller, schedule virtual machines deployment on node and manages the virtual networks.
- Node controller, manages starting, querying shutting-down of virtual machines on the physical machines.
- Storage controller, manages block storage providing the same functions as Amazon Elastic Block Storage.
- Walrus, manages persistent storage of data, organized in buckets and objects, compatible with Amazon S3.

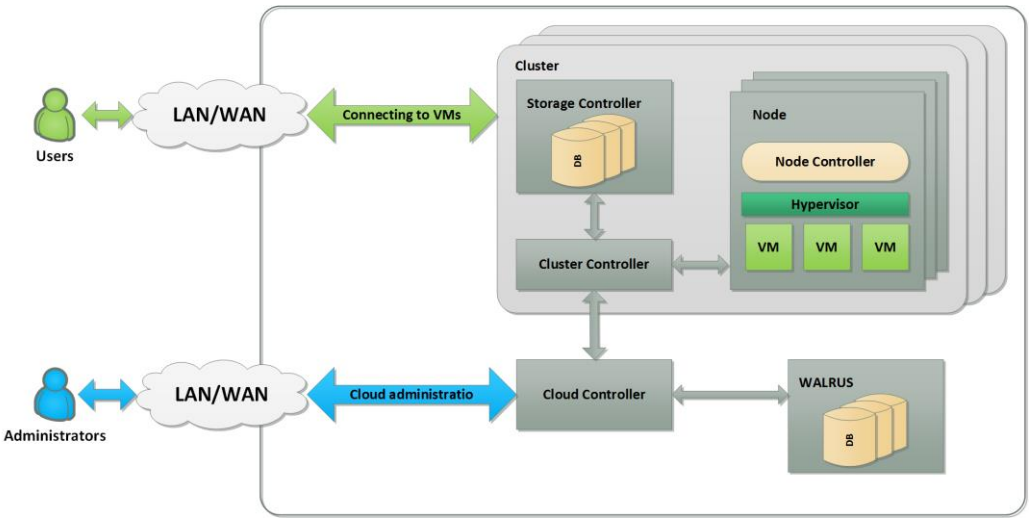

Fig. 7. Eucalyptus Basic Architecture

Cloud controller is a collection of web services exposing cloud administration functions. It is the entry point for the whole Eucalyptus system. It also implements a Web interface for cloud's services management. These services are grouped in:

- Resource services, processing requests related to virtual machines, accepting or rejecting them based on the global system status. Service Level Agreements are monitored and enforced at this level.
- Data services, manage creation and retrieval of user and system generated data.
- Interface services allow user access to web interfaces and cloud services. It provides interfaces compatible with Amazon Web Services.

Cluster controller manages cloud nodes (physical machines). It schedules and dispatches virtual machine instance creation requests to the nodes, it manages the virtual network overlay. Data about node status is gathered and aggregated at this level. By default, the cluster controller is executed on a different server than the nodes and it requires direct connection also with the cloud controller.

The *Node controller* is executed on each server designated to execute virtual machines. It gathers information regarding the hardware configuration of each machine on which is run and report them to the cluster controller. The node controllers do not start or stop the virtual machines, this being the function of the cluster controller.

DOI: 10.12948/issn14531305/19.4.2015.06

Storage controllers manage network block devices. Exported volumes (elastic block storages) can be attached to virtual machines but cannot be shared between multiple instances. Data stored on these volumes is persistent after virtual machines are stopped. Snapshots of the volumes can be created and stored using Walrus.

Walrus is a put/get service for storing persistent objects. It is compatible with Amazon S3 and provides SOAP and REST interfaces to its functions. Eucalyptus uses Walrus for virtual machine image storing.

Standards

Eucalyptus was designed from the start as Amazon Web Service compatible, it implements large parts (but not all) of EC2 API through Euca2ools, a set of command line tools, and S3 API through Walrus component. There is no OCCI API support though an implementation was defined as a target of "Flexible Services for the Support of Research", a 2010 project, without a palpable outcome. There is no CDMI or OVF support/implementation for Eucalyptus. While not providing native support, Apache DeltaCloud has a Eucalyptus driver giving it indirect access to interfaces using CIMI API.

11 OpenNebula

OpenNebula was designed to offer a simple and flexible solution offering a complete set of functionalities for installing and managing enterprise clouds and virtualized data centers [19]. Started in 2006 as a research project, its first version was published in 2008, OpenNebula is now an open-source project.

OpenNebula offers four type of interfaces for interacting with and administering the cloud:

- For the consumers of cloud resources, there is an interactive web interface as well as APIs compatible with Amazon Web Services EC2, EBS and S3.
- Administrators have access to a full set of command line utilities as well as to a dedicated web interface.
- Service integrators can use low level APIs written in Ruby, Java and XMLRPC API.
- A catalogue of third party appliances, ready to be used in the OpenNebula environments.

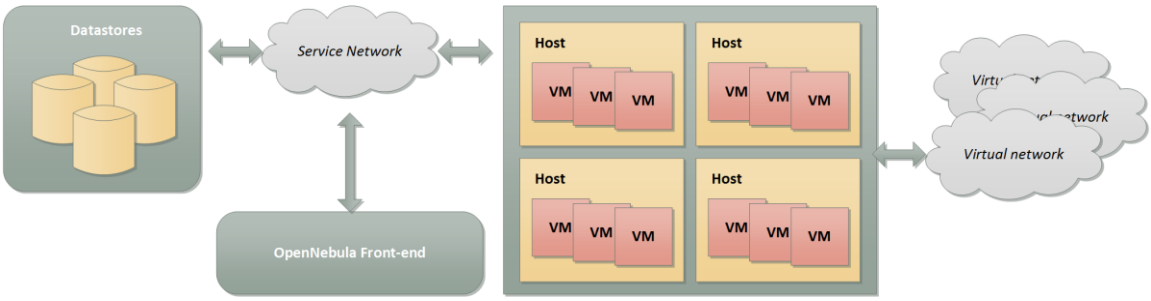

Fig. 8. OpenNebula Architecture

OpenNebula has a modular architecture Figure 8 offering the possibility to use off-the-shelf as well as enterprise grade hardware and software components as hypervisors, for monitoring, storage or network communications. Any OpenNebula deployment will feature:

- A front-end executing the OpenNebula services.
- Hosts with hypervisors providing the resources for executing virtual machines.
- Data stores for the virtual machines images.
- Physical networks linking the cloud components.

The front end is a server having OpenNebula installed on it. It is connected with all the cloud's hosts. The management daemons, the scheduler and the administration web interface are running on this machine.

Any deployment consists of at least a zone Figure 9: a group of interconnected physical hosts with hypervisors controlled by OpenNebula [20], typically no more than 500 in a zone. More than one zone can be managed using OpenNebula oZones component and

DOI: 10.12948/issn14531305/19.4.2015.06

can be combined to form a Virtual Data center (VDC). Using zones guarantees complete users and domains isolation, increased scalability, centralized management [21].

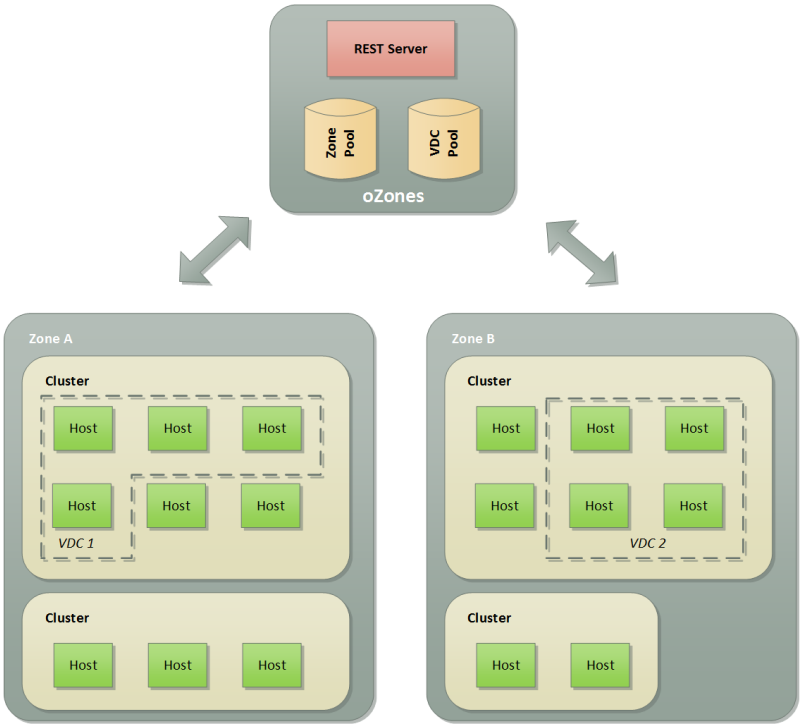

Fig. 9. OpenNebula Zones Architecture

Hosts are physical machines on which virtual machines are executed. A supported hypervisor must be installed on them (Xen, KVM or VMware) under one of the operating systems certified for OpenNebula (RedHat Enterprise Linux, Ubuntu Server, SUSE Linux Enterprise, CentOS, openSUSE, Debian.

Virtual machine images are managed through datastores, usually a SAN/NAS, always available to the front-end server. System datastores are those used for running virtual machine images. Image datastores store disk images. These are copied/cloned from/to the system datastores when virtual machines are started/stopped or when snapshots are taken. An image datastore can be [22]:

- A filesystem, when the images are stores as files on volumes mounted from a NAS/SAN.
- Vmfs, specialized datastore using VMFS format for the use or VMware hypervisors. It is not UNIX-comptabile and cannot be mounted on the front-end server.
- LVM: LVM volumes can be used instead of file systems
- Ceph, specialized for use with Ceph block devices.

Interfaces

OpenNebula was designed as an expandable, modular system, allowing deployment of customized Cloud Computing architecture and easy interaction with data-center services. Its interfaces are subdivided as cloud interfaces, targeted against cloud resource consumers, and system interfaces (**Fig. 10**). Cloud interfaces provide an abstraction layer above OpenNebula's services, allowing software tools and components to be build for cloud interaction. System interfaces provide access to all the cloud's services and are used to adapt the cloud to the targeted infrastructure.

DOI: 10.12948/issn14531305/19.4.2015.06

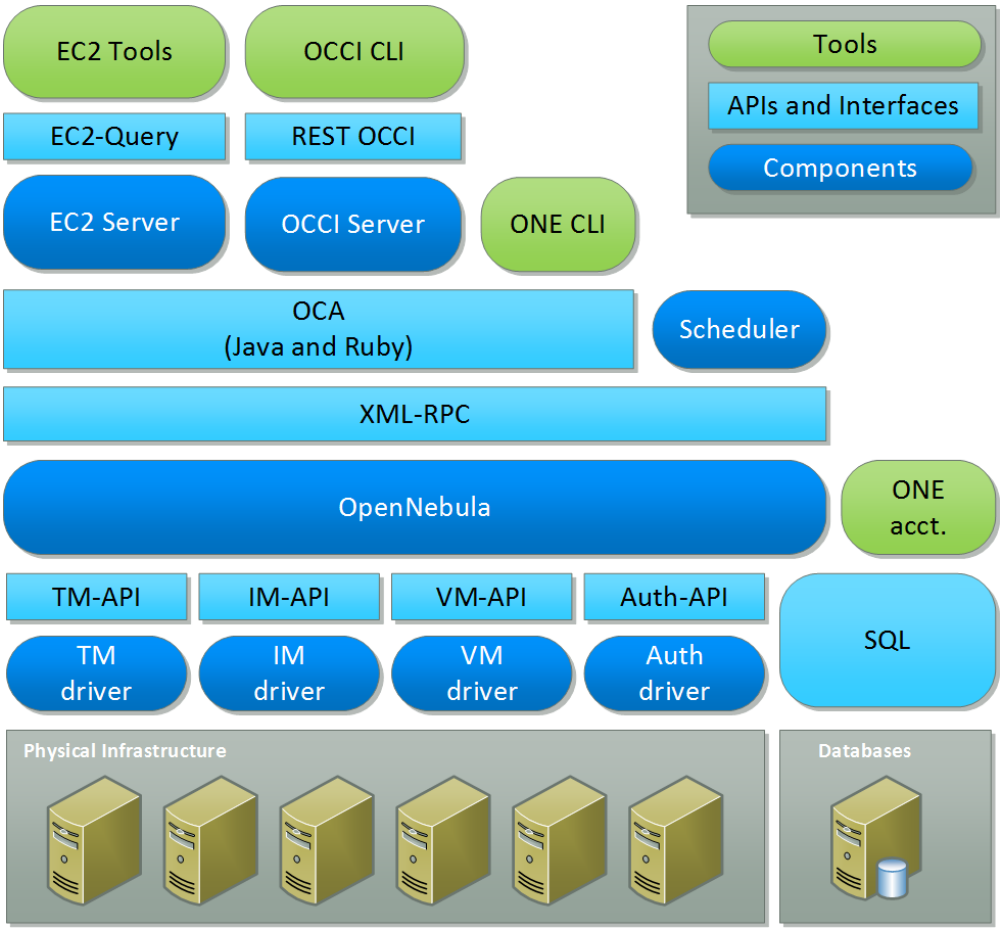

Fig. 10. OpenNebula Interfaces, adapted after [23]

Cloud interfaces manage virtual machines, networks and images using standard APIs such as AWS EC2 and OGF OCCI. System interfaces use XML-RPC or OpenNebula's own API, having bindings for Ruby and Java programming languages.

Standards

As all the other open source Infrastructure as a Service platforms, OpenNebula implements the de facto standard: Amazon Web Services EC2 and S3 APIs. OCCI has a native implementation through a pluggable component and another one by using the rOCCI project. OVF is supported by using a separate Java component. No direct CIMI interface is available but Apache Deltacloud provides a driver for OpenNebula using its OCCI server component.

12 Conclusion

As the Cloud Computing industry matures, the Infrastructure as a Service platforms tend to implement a homogenized set of functions. Choosing a platform might depend on its openness and adherence to standards, exposing the cloud or the appliance to the greatest number of clients. While implemented open standards might look as a sure way to achieve this goal, in practice, even the major open-source platforms offer limited support in this area. Portability and interoperability can only be assured for now by using the de facto standard, Amazon Web Services APIs. Putting Apache Deltacloud aside, there is no native or third party support for Cloud Infrastructure Management Interface on any of the studied IaaS platforms. At the same time Apache Deltacloud adds its own API and cannot ignore AWS EC2/S3 compatibility.

DOI: 10.12948/issn14531305/19.4.2015.06

Table 1. Main standards support in major open source IaaS platforms.

Standard Platform	**OCCI**	**CDMI**	**CIMI**	**OVF**	**AWS**	**Apache Deltacloud driver**
OpenStack	3rd party component	3rd party component for CDMI 1.0.2	Apache Deltacloud	-	EC2, S3	Yes
Apache CloudStack	rOCCI	-	-	-	EC2, S3, 60%	No
OpenNebula	rOCCI, native OCCI 1.1	-	Apache Deltacloud through OCCI	3rd party Java component	EC2, S3	Yes
Eucalyptus	-	-	Apache Deltacloud	-	EC2, S3	Yes

Cloud Standards Coordination, an initiative launched by the European Commission and The European Telecommunications Standards Institute, identified in an October 2015 report no less than 16 standardization organizations and 114 documents related to cloud standards (94 with the status “Published”, 14 “Draft” and 6 “In Progress”) [24]. None of these are likely to enjoy in the near future the massive adoption displayed by AWS APIs but lessons from it could and are learned and, as the Cloud enabled applications become ubiquitous, we can only hope that an open, vendor neutral, Cloud interfacing specification gains more traction and becomes both a *de facto* and a *de jure* standard.

References

[1] "Open Cloud Computing Interface | Open Standard | Open Community," [Online]. Available: http://occi-wg.org/. [Accessed 2015].

[2] "Open Cloud Computing Interface - Core," 2011.

[3] T. Metsch and A. Edmonds, "OGF Published Documents," June 2011. [Online]. Available: http://www.ogf.org/documents/GFD.185.pdf.

[4] T. Metsch and A. Edmonds, "Open Cloud Computing Interface - Infrastructure," 2011.

[5] A. Edmonds, T. Metsch, A. Papaspyrou and A. Richardson, "Open Cloud Computing Interface: Open Community Leading Cloud Standards," [Online]. Available: http://ercim-news.ercim.eu/en83/special/open-cloud-computing-interface-open-community-leading-cloud-standards. [Accessed 2015].

[6] S. Fiore and G. Aloisio, Grid and Cloud Database Management, Springer-Verlag Berlin Heidelberg, 2011, pp. 25,26.

[7] "Cloud Data Management Interface (CDMI)," [Online]. Available: http://www.snia.org/cdmi. [Accessed 2015].

[8] "Cloud Infrastructure Management Interface Model and RESTful HTTP-based Protocol," 2013.

[9] VMware, "The Open Virtual Machine Format Whitepapre for OVF Specification," 2007. [Online]. Available: https://www.vmware.com/pdf/ovf_whitepaper_specification.pdf.

[10] "Deltacloud drivers," [Online]. Available: http://deltacloud.apache.org/drivers.html#drivers. [Accessed October 2015].

[11] J. Atul, D. Johnson, M. Kiran, R. Murthy, C. Vivek and G. Yogesh, OpenStack Beginner's Guide, CSS Corp, 2012.

[12] K. Pepple, Deploying OpenStack, O'Reilly Media, 2011, p. 88.

DOI: 10.12948/issn14531305/19.4.2015.06

[13] "Apache CloudStack: Open Source Cloud Computing," [Online]. Available: https://cloudstack.apache.org/. [Accessed 5 2015].
[14] N. Sabharwal and R. Shankar, Apache CloudStack Cloud Computing, Birmingham: Packt Publishing, 2013, p. 24.
[15] D. Nalley and S. Eizadi, "Development 101," 2013. [Online]. Available: https://cwiki.apache.org/confluence/display/CLOUDSTACK/Development+101.
[16] "Programmer Guide - Apache CloudStack 4.5.1. documentation," 6 2015. [Online]. Available: http://docs.cloudstack.apache.org/en/latest/dev.html.
[17] D. Nalley, "Apache CloudStack and the cloud API wars," December 2013. [Online]. Available: http://www.comparethecloud.net/opinions/apache-cloudstack-and-the-cloud-api-wars/.
[18] Y. Wadia, "The Eucalyptus Open-Source Private Cloud," Cloudbook, vol. 3, no. 1, pp. 29-32, 2012.
[19] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov, "The Eucalyptus Open-source Cloud-computing System," in Proceedings of 2009 ACM/IEEE International Conference on Grid Computing, 2009.
[20] "An Overview of OpenNebula - OpenNebula 4.12.1 documentation," 5 2015. [Online]. Available: http://docs.opennebula.org/4.12/design_and_installation/building_your_cloud/intro.html.
[21] "OpenNebula Documentation," [Online]. Available: http://archives.opennebula.org/documentation:archives:rel3.4:zonesmngt. [Accessed 2015].
[22] "OpenNebula Zones Overview 4.4," 4 2015. [Online]. Available: http://archives.opennebula.org/documentation:rel4.4:ozones.
[23] "OpenNebula Storage Overview 4.4," [Online]. Available: http://archives.opennebula.org/documentation:rel4.4:sm. [Accessed 2015].
[24] "OpenNebula," [Online]. Available: http://archives.opennebula.org/documentation:archives:rel4.2:introapis. [Accessed 2015].
[25] "Cloud Standards Coordination - A deeper look at Cloud Computing," October 2015. [Online]. Available: http://csc.etsi.org/phase2/snapshot2/StandardsOrganizations.html.

Andrei IONESCU obtained his Master of Science diploma in Economic Informatics in 2014. He has more than 10 years of experience as a software developer, currently working as an independent contractor. Since 2014 he is a PhD candidate at the Bucharest Unviersity of Economic Studies with a thesis in the field of Resource Management in Cloud Computing.

DOI: 10.12948/issn14531305/19.4.2015.06