Towards A Standards-Based Cloud Service Manager

Amine Ghrab[1], Sabri Skhiri[1], Hervé Kœner[2] and Guy Leduc[2]
[1]*Eura Nova R&D, Mont-Saint-Guibert, Belgium*
[2] *Electrical Engineering and Computer Science Department, University of Liège, Belgium*
{amine.ghrab, sabri.skhiri}@euranova.eu, herve.koener@student.ulg.ac.be, guy.leduc@ulg.ac.be

Keywords: Cloud Computing, OCCI, Service Management

Abstract: Migrating services to the cloud brings all the benefits of elasticity, scalability and cost-cutting. However, migrating services among different cloud infrastructures or outside of the cloud is not an obvious task. In addition, distributing services among multiple cloud providers, or on a hybrid installation requires a custom implementation effort that must be repeated at each infrastructure change. This situation raises the lock-in problem and discourages cloud adoption. Cloud computing open standards were designed to face this situation and to bring interoperability and portability to cloud environments. However, they target isolated resources, and do not take into account the notion of complete services. In this paper, we introduce an extension to OCCI, a cloud computing open standard, in order to support complete service definition and management automation. We support this proposal with an open-source framework for service management through compliant cloud infrastructures.

1 INTRODUCTION

Cloud computing provides an infrastructure emphasizing resources multiplexing, data locality, and elasticity. The offerings present new service deployment models to meet the needs of customers, switching the focus from the infrastructure management details to the core business. However, cloud computing still presents multiple drawbacks, mainly interoperability, portability and security issues. This has lead to a situation where cloud offerings are considered as isolated islands with no transport channels between them. Customers face the threat of having their deployed services locked into a single cloud provider.

To solve these issues, we operated at the lowest level, Infrastructure as a Service. We gathered the basic requirements, mainly clear representations of the resources along with their compliant implementations and a single management point to handle the different API. Cloud computing open standards already provide such representations. We selected Open Cloud Computing Interface (OCCI)[1] for reasons we detail in section 3. Then, we based our work on compliant implementations developed by OpenNebula[2] and OpenStack[3].

In this paper we propose two contributions, (1) we extend the OCCI standard to support cloud service managers, and (2) we provide a reference implementation as an open-source tool[4] to seamlessly manage the services deployed within standards-based cloud offerings.

The paper is organized in five parts. Section 2 explains cloud concepts and technologies with a focus on the Infrastructure as a Service level and addresses the current open issues to deal with. To solve the above-mentioned issues, Section 3 provides a walk through existing solutions, and analyses their efficiency and usefulness for later research. Section 4 introduces our approach based on cloud computing open standards and highlights the design and implementation of our solution. Section 5 proposes future research directions to tackle the remaining issues. Section 6 describes related works and how our solution goes beyond the state of the art.

2 CLOUD COMPUTING

Cloud computing has changed the way people are perceiving information technology. It helps focusing more on the core activities rather than managing all

[1] http://occi-wg.org
[2] http://opennebula.org/
[3] http://www.rackspace.com/
[4] https://github.com/KoenerHerve/Service-Manager

the service delivery stack starting from the underlying hardware infrastructure.
This makes Cloud Computing paradigm the driver of information technology for the years to come. For the best of our knowledge, the most accepted definition of cloud computing is the one introduced by the National Institute of Standards and Technology (NIST) and that summarizes cloud computing as: "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell and Grance, 2011).

2.1 Service Models

According to the abstraction level and the user control over the services, cloud offerings could be divided into three levels:

- Infrastructure as a Service (IaaS): The lowest level of cloud offering stack. Resources are mainly virtual machines, virtual disks and virtual networks. There are two types of IaaS model, hosted model such as Amazon[5] offerings, and on-premise model such as OpenNebula.
- Platform as a Service (PaaS): The intermediate level of the stack. Customers have access to a development environment where they can host, develop and deploy their applications. Control is then switched to the level of the application environment and its related data. Google AppEngine[6] and Heroku[7] are examples of such services.
- Software as a Service (SaaS): The top level of the cloud stack. Access is limited to preconfigured software without control over the underlying platform or hardware such as Google Apps[8] or Salesforce[9].

2.2 Architecture

Cloud computing is the fruit of a convergence of many technologies, especially distributed systems. Multiple cloud architectures were introduced to define cloud computing levels, actors and communication modes. In the literature, several reference cloud architectures were proposed including Reservoir (Rochwerger et al., 2009), NIST (Bohn et al., 2011) and IBM (Behrendt et al., 2011).
Among these architectures, Reservoir proposes a well defined architecture as well as an open source implementation. We have then chosen it as the architectural background for this work. The purpose of Reservoir is to "design a cloud computing architecture from basic principles where each provider is an autonomous business, federating with others, and the management of sites is governed by policies aligned with the site's business goals" (Rochwerger et al., 2009).
Reservoir (Fig. 1) organizes the IaaS service model into three levels :

- Virtual Execution Environment Host (VEEH): provides basic control and monitoring of VEEs (generalization of the Virtual Machine concept) life cycle. VEEHs could be hypervisors such as KVM or Virtual Java Service Containers (VJSC), a technology that enables Java applications deployment in virtual containers and their migration between hosts.
- Virtual Execution Environment Manager (VEEM): ensures optimal placement of VEEs into VEEHs and federation of remote sites. OpenNebula and OpenStack are the ones we cover in this paper.
- Service Manager: handles the deployment, provisioning and Service Level Agreement(SLA) compliance enforcement of services, defined as sets of related VEEs. RightScale[10] and AWS CloudFormation[11] being proprietary, we built our own service manager described later.
- Layers of interoperability: ensure Vertical and horizontal interoperability between layers using open standards.

In this paper we focus on service manager level. Service manager is an abstraction layer on top of the VEEM. It helps avoiding the take it or leave it aspect of current cloud offerings. Therefore, a service manager provides a common interface to manage the complete service's life-cycle.

While VEEMs free users from direct physical infrastructure management by offering virtualized resources, the use of service managers helps users go from individual VM management to whole services management, thus focusing more on business goals.

To better understand the service manager role, we take a common scenario of a company deploying its service in a traditional way, which takes it through the following steps:

[5] http://aws.amazon.com
[6] http://code.google.com/appengine/
[7] http://www.heroku.com
[8] www.google.com/apps
[9] http://www.salesforce.com/
[10] http://www.rightscale.com
[11] http://aws.amazon.com/cloudformation/

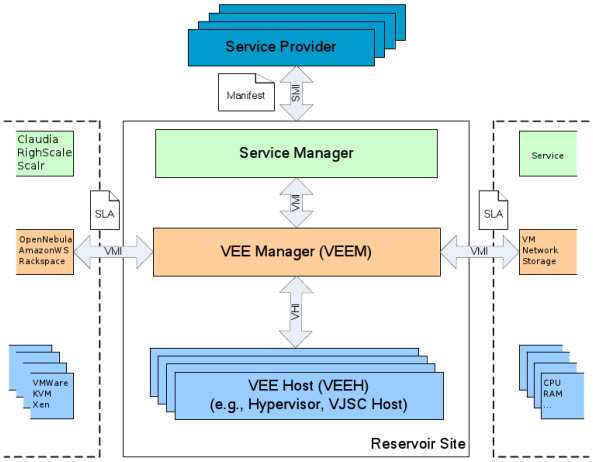

Figure 1: Reservoir Architecture (Rochwerger et al., 2009)

1. Define the service topology (architecture and resources) and lease dedicated or shared servers with over-provisioning in mind to meet load peaks.
2. Manually install and configure the service parts on each of the leased servers.
3. Implement failover and security measures.
4. Launch the service.
5. Monitor the service.
6. Provision and release resources to meet the load and perform infrastructure configuration routine.

These tasks require a lot of manual system administration to setup and to ensure healthy running of the overall service. However, with large scale systems, such tasks would require a lot of investment, both human and financial, to keep the promised QoS. Startups and small companies could not afford that and may become victims of their own success.

IT service management presents repetitiveness and similarity in most of the sub-tasks. This makes it a potential candidate to automation and cloud deployment. The latter leverages the service owner from resources over-provisioning through the on-demand and pay-as-you-go consumption mode. Automation could be achieved using cloud computing service managers that abstract away the direct infrastructure resources management, and instead allows for services applications management.

The service manager should take care of:

- Instances provisioning: providing a transparent provisioning of resources from multiple clouds in a multi-IaaS sourcing fashion.
- Configuration Setup: automating the configuration of newly provisioned instances based on service descriptions.
- SLA compliance: tools for the monitoring of the deployed services which could then auto-scale to dynamically follow load picks with minimal management effort. Failover mechanisms should also be implemented to continue operating the services after components failure.

Using service managers, the service deployment process would consist of defining the service topology and the runtime logic, i.e. scaling rules and failover measures.

2.3 Open issues

When building a cloud computing infrastructure, architects struggle to satisfy both requirements from usage and deployment scenarios and constraints from existing infrastructure and processes. Cloud computing still presents many challenges, mainly interoperability, portability and security issues.

2.3.1 Interoperability

Within the cloud context, A. Edmonds(2011) defines interoperability as follows: "To be interoperable means to imbue the common abilities of mobility to cloud service instances, to extract all service instance described by a common representation, to share all cloud service instance related data in and out of providers and to allow cloud service instances to work together". [12]

The purpose is to enable different services deployed on different cloud infrastructures to communicate and cooperate on the fly. Interactions between resources is made through common interfaces. These same interfaces are to be used by the service manager without the need to provide a specific adapter for each separate API.

2.3.2 Portability

Portability is the ability to provide a common format by which services are described, deployed, retrieved and reinserted with no modifications independently from the provider. This includes API, workload and data portability. Service portability is the key to overcome cloud vendor lock-in.

A key difference between portability and interoperability is that a direct link between systems is needed for the latter but not the former.

These challenges are root issues that gave birth to subsequent ones such as federation, dynamic work-

[12] http://andy.edmonds.be/blog/2011/06/13/cloud-interoperability/

load distribution among providers, services migration, and as a result the lock-in problem.

3 OPEN STANDARDS

In an interoperable cloud ecosystem, resources should have a commonly agreed representation and managed through standard APIs. The related data would have a common format which allows different services from different cloud providers to manage it. The similarity of IaaS offerings makes this level the most likely starting point to solve the lock-in issue. A possible solution is to provide standardized representations that cover service lifecycle management along with its underlying computing, network and data resources.

3.1 Choice of the standards

Cloud providers APIs are unique and platform-specific with no interoperability mechanisms between them. An approach to tackle this issue is by providing a specific driver per API, as implemented by Rightscale and Scalr. Deltacloud[13] is providing a single entry point to developers, while maintaining a driver per API backend.
Both approaches are interesting and have been implemented. However, they remain dependent on the providers proprietary APIs. They have to adapt their drivers to meet each change on these APIs. Moreover, they introduce an additional layer to the architecture which might increase latency.

To deal with this situation, we chosen to rely on cloud open standards. Here the API maintenance is handled by the cloud provider that comply with the standards. Our purpose is to write a standards compliant client and maintain it according to these standards.

The Cloud Standard Coordination[14] provides a comprehensive list of cloud computing standards. In this paper, we focus only on open standards that provide answers to interoperability and portability issues. According to the NIST(Hogan et al., 2011), only Open Virtualization Format(OVF)[15] has achieved market acceptance in system portability. The Cloud Data Management Interface (CDMI)[16] and OCCI are considered as market approved standards for service interoperability and data portability. Other standards such as IEEE Cloud Portability and Interoperability Profiles (CPIP) or IEEE Standard for Intercloud Interoperability and Federation (SIIF) are still under-development.

3.2 OCCI

OCCI is an initiative from the Open Grid Forum (OGF)[17] to provide a standard RESTful Protocol and API for the management of cloud resources. It seeks covering the three service models, we target the IaaS level.

OCCI provides a cloud management interface with focus on interoperability. It enables the interoperability and the integration between different IaaS by providing common mechanisms for the representation, discovery and management of the underlying resources.

Resources representation is based on a type system, described in the core model document of the specification (Metsch et al., 2010c). Within this model, a hierarchy governs cloud resources, which eases their classification and thereof their identification. The central type of the OCCI model is *Resource* which represents the resources offered by the cloud provider. Resources relationships are modeled using *Link*. Resource and Link inherits both from Entity, which is an abstract class complementing Category. *Category* is used for uniquely typing and identifying resources within the underlying system. Category is a superclass for Kind and Mixin. *Kind* defines resource characteristics and serves as the base of the classification system. *Mixin* enables dynamic addition and removal of capabilities from the resource. *Action* is attached to mixins and kinds and denotes the operations a resource could perform.

In addition to classification and management, OCCI provides mechanisms to discover supported resources and operations. Such a feature is critical to automate cloud management and to follow the dynamically changing nature of resources properties.

An extension of the core model to cover IaaS level is depicted in the infrastructure model (Metsch et al., 2010a). The extensibility of OCCI enables to represent a wide range of resources and perform complex operations on them, such as management of databases and key-value stores (Edmonds et al., 2011a).

The third document released by OGF is the OCCI HTTP specification (Metsch et al., 2010b). Resources in this representation are identified and accessed using unique Uniform Resource Identifiers(URI). As it is a RESTful API, the management of resources is done through the basic HTTP operations POST, GET, PUT and DELETE that perform Create, Retrieve, Update

[13] http://deltacloud.apache.org/
[14] http://cloud-standards.org
[15] http://www.dmtf.org/standards/ovf
[16] http://www.snia.org/cdmi
[17] http://www.ogf.org/

and Delete(CRUD) operations on the underlying resources.

The purpose of OCCI is not to cover all possible cloud management tasks, instead it offers a common infrastructure on top of which cloud offerings could be built (Edmonds et al., 2011b). Obviously, the set of supported operations is limited compared to those offered by specific cloud offerings. However, extensions to OCCI or direct use of the providers API remains as available options.

OCCI is already supported by many cloud management platforms, including OpenNebula and OpenStack(Edmonds et al., 2012).

3.3 Standards Integration

A key point in the choice of OCCI, in addition to being open and accepted, is their integration. A common use case is to have a service deployed across multiple virtual machines. Each have its attached storage volume, and the group is related with a set of networks.

OVF is a DMTF standard for packaging, distributing and deploying virtualized resources. It is used for Virtual Appliances, pre-configured software applications installed on a set of virtual machines. The packaged virtual resources act as templates or plug and play services. OVF virtual appliances have the advantage of platform independence, i.e. the same virtual resource could be deployed on a Xen or VMware Hypervisor. This enables moving VMs between cloud providers. OVF has also been used to describe services deployment and operation on the cloud as demonstrated on (Galán et al., 2009), for example to specify minimum requirements and scaling conditions of the service.

CDMI is another open standard developed by SNIA[18]. It provides a data packaging format to describe the data, its metadata and the operations to fulfill packaging and transferring tasks through the cloud. CDMI serves as an interface to perform CRUD operations and enables users to discover and manage individual or sets of data elements. Data management related aspects such as security access and billing policies are also covered by this standard.

Put together, OVF provides a format to package the virtual machines in a common portable format independently from the hypervisor. CDMI encapsulates storage resources within containers ready to be exported and attached to virtual machines. OCCI completes the picture by providing a common interface that handles these packaged virtual machines and containers. Such configuration supports both common management tasks as well as specific scenarios such as cross-clouds migration.

[18] http://www.snia.org/

4 OCCI SERVICE MANAGER

4.1 OCCI Extension

As mentioned earlier, a service manager should provide a uniform management interface to handle the deployment and the monitoring of whole interrelated entities while encapsulating low-level management details. OCCI being our chosen standard, we suggest an extension to the core specification on top of which we built our service manager. The extension introduces the concepts related to cloud IaaS service management such as Service, Role, Group and dependency (see Fig. 2):

1. Role: A configured virtual machine fulfilling a specific functionality, such as a DBMS.
2. Service: A set of interconnected roles serving a determined goal, such as web portal.
3. Group: A set of roles with the same initial configuration. The elasticity aspect of the service is reflected by adding and removing group members.
4. Dependency: Describe an order relationship between roles. If role A needs a role B to be available before being able to start, then A is called dependent on B.

A typical use case is a web service which could be defined as a set of interdependent roles that the user selects, classifies within groups and defines their dependencies. Figure 3 depicts such scenario.

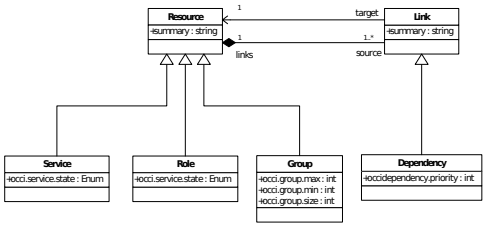

Figure 2: Service Manager Class Diagram

4.2 Implementation

We have implemented OCCI Service Manager [19] as a proof-of-concept for our extension to the OCCI model. The framework allows the user to define the

[19] https://github.com/xxx

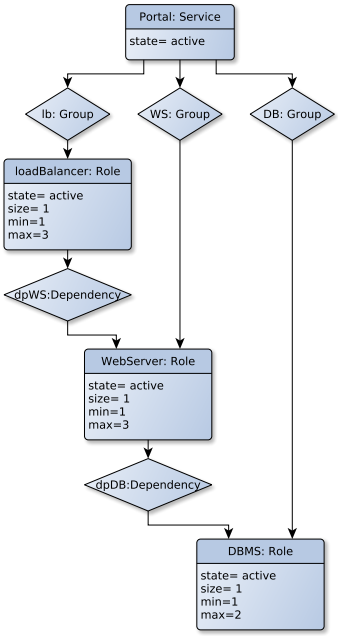

Figure 3: Service Deployment Example

service, to start it and to dynamically allocate underlying resources. The descriptions of the resources and the users profiles are stored in a MySQL database. The OCCI Service Manager offers a web interface and a RESTful API to enable users to start, stop virtual machines and to retrieve different monitoring information.

The framework is written in Ruby and built using the Sinatra[20] light-weight framework. It stands as a middleware between end-users and the OpenNebula OCCI server.

The first step to define a service is to define the roles which compose it. To create a role, the user specifies the OS template and links it to a subsequent mixins. Mixins allow the user to specify the dynamic add-ons of the role. For instance, we can specify that the virtual machines of a certain role uses CentOS template and acts as an Nginx load balancer.

The second step is to define the different dependencies between roles. For instance, a load balancer depends on web server roles before it could be started.

When the user wants to start a service, the application starts all the virtual machines of the roles defined for the service. If the virtual machines of a certain role have dependencies the OCCI Service Manager starts first the dependencies.

When a virtual machine is started, the OCCI Service Manager configures it automatically according to the mixins linked to the role of this virtual machine. A service is started on each of the virtual machines and acts as an agent allowing the execution of remote commands. Once all virtual machines are started and configured the state of the service is set to running.

The user can perform CRUD operations on the service or on the roles which compose it. He can also change the behavior of the virtual machines of a certain role by linking it to another Mixin at runtime. The roles provisioning process is depicted on Figure 4.

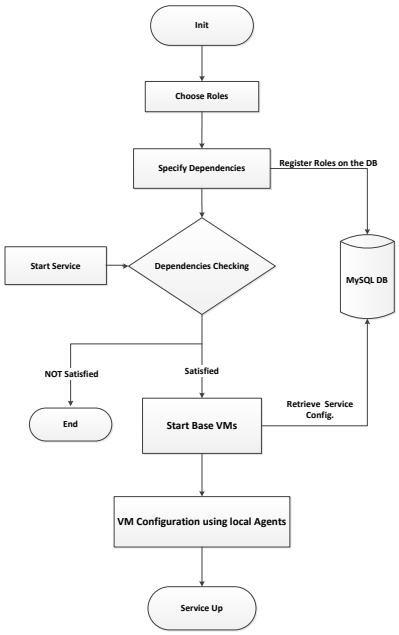

Figure 4: Service provisioning process

5 FUTURE WORKS

A service manager within the Reservoir model is assigned two main tasks. The first one is resources provisioning and deployment, which our service manager already supports. The second one is monitoring and enforcing SLA compliance. Currently, we do not include monitoring modules to follow the load on the provisioned resources. We envision integrating Ganglia[21] as our distributed monitoring system. Ganglia is supported by OpenNebula[22].

Scheduling is an interesting feature that helps optimize resource utilization. Haizea[23] is part of the

[20] http://www.sinatrarb.com/

[21] http://ganglia.sourceforge.net/

[22] http://opennebula.org/documentation:rel3.6:ganglia

[23] http://opennebula.org/software:ecosystem:haizea

OpenNebula ecosystem and is a potential candidate to integrate scheduling within our service manager.

Accounting and billing is another responsibility of a service manager. Users are granted permissions, follow their consumption and control their budget through this features. We plan to integrate this component with a friendly user interface for better visualization.

We are also considering enriching the service manager with further features such as full service cloning and snapshotting.

6 RELATED WORKS

A plethora of projects were initiated with the goal of implementing services management tools. Here we cover some of the open-source projects aligned with our work.

Claudia was developed by Telefonica R&D as part of the Seventh Framework Programme of the European Union(FP7). It is a service manager supporting lifecycle management, configuration, scaling and monitoring of services deployed across multiple cloud providers (Moran, 2012). The structure of the service is described using an extension of OVF (Rodero-Merino et al., 2010). Claudia implements a driver per API approach to support different cloud providers. In addition, it also provides an OCCI implementation to support compliant interfaces. We consider that since OCCI is designed for service management, it fits natively better as a service descriptor than OVF. Unfortunately, Claudia is no longer supported as an open-source project[24].

HP IaaS Aggregator is developed by HP for the purpose of providing common interface and description of IaaS services across multiple IaaS service providers (Lee et al., 2011). It's based on the Common Information Model(CIM)[25] standard, a DMTF model to describe entities and their attributes. The IaaS Aggregator is a cloud management console fulfilling tasks such as resources lifecycle and security management. It is based on a Web2Exchange proxy and communications are made through a REST interface. The main difference between our proposal and the IaaS Aggregator is the choice of standard. OCCI is built from the ground up to support resources management on the cloud, while CIM is a general purpose standard for representing resources as a set of objects and their relationships. According to the class diagram depicted on the IaaS Aggregator project paper (Lee et al., 2011), the IaaS Aggregator is similar to the infrastructure extension of the core OCCI model. Furthermore, network topology descriptions are not very well supported by CIM (Ghijsen et al., 2012).

Scalr is a cloud management tool supporting multiple cloud providers while adopting the classic approach of a specific driver per API. Scalr is comprised basically of a central PHP management console and a Python agent installed on the remote instances. As an alternative to the proprietary RightScale, it supports services lifecycle management such as auto-scaling, auto-configuration and fail-over. The VM management process in Scalr inspired us when dealing with the technical details of the implementation. The main bottleneck of Scalr is its adoption of the driver-per-API approach explained in Section 3. A possible enhancement to Scalr is to equip it with an OCCI driver.

7 CONCLUSIONS

In this paper, we have proposed our contribution in which we attempt to advance the state of the art of cloud computing service management.

Currently customers who plan to migrate their services to the cloud face the threat of being locked-in. As interoperability and portability are traditional drivers for standards development, we decided to refer to cloud computing open standards as the backbone to solve these problems.

Targeting cloud computing end-users, we built our solution as an open-source service manager. We have first extended the OCCI standard to support service management. We then implemented the solution. The result is a tool that abstracts the underlying vendors APIs details. It helps users deploy and manage their services seamlessly among open standards compliant clouds.

The project is in progress, further works needs to be done in order to provide a complete solution that covers advanced cloud service management features. On the open standards side, specifications need to be enriched in order to cover more cloud functionalities and gain market acceptance.

Acknowledgment

We wish to thank Salim Jouili for his comments on the draft.

[24] https://lists.morfeoproject.org/pipermail/claudia-users/

[25] http://dmtf.org/standards/cim

REFERENCES

Behrendt, M., Glasner, B., Kopp, P., Dieckmann, R., Breiter, G., Pappe, S., Kreger, H., and Arsanjani, A. (2011). Introduction and architecture overview ibm cloud computing reference architecture 2.0. *Draft Version V*, 1:0.

Bohn, R. B., Messina, J., Liu, F., Tong, J., and Mao, J. (2011). Nist cloud computing reference architecture. In *Proceedings of the 2011 IEEE World Congress on Services*, SERVICES '11, pages 594–596, Washington, DC, USA. IEEE Computer Society.

Edmonds, A., Metsch, T., and Papaspyrou, A. (2011a). Open cloud computing interface in data management-related setups. volume 1, pages 23–48. Springer.

Edmonds, A., Metsch, T., and Papaspyrou, A. (2011b). *Open Cloud Computing Interface in Data Management-Related Setups*, volume 1, pages 23–48. Springer.

Edmonds, A., Metsch, T., Papaspyrou, A., and Richardson, A. (2012). Toward an open cloud standard. *Internet Computing, IEEE*, 16(4):15–25.

Galán, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., and Vaquero, L. (2009). Service specification in cloud environments based on extensions to open standards. In *Proceedings of the fourth international ICST conference on communication system software and middleware*, page 19. ACM.

Ghijsen, M., van der Ham, J., Grosso, P., and de Laat, C. (2012). Towards an infrastructure description language for modeling computing infrastructures. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 207–214. IEEE.

Hogan, M., Liu, F., Sokol, A., and Tong, J. (2011). Nist cloud computing standards roadmap. *NIST Special Publication*, page 35.

Lee, B., Yan, S., Ma, D., and Zhao, G. (2011). Aggregating iaas service. In *SRII Global Conference (SRII), 2011 Annual*, pages 335–338. IEEE.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. *NIST special publication*, 800:145.

Metsch, T., Edmonds, A., et al. (2010a). Open cloud computing interface–infrastructure. In *Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN)*.

Metsch, T., Edmonds, A., et al. (2010b). Open cloud computing interface–restful http rendering. In *Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN)*.

Metsch, T., Edmonds, A., and Nyrén, R. (2010c). Open cloud computing interface–core. In *Open Grid Forum, OCCI-WG, Specification Document. Available at: http://forge. gridforum. org/sf/go/doc16161*.

Moran, D. (2012). *Claudia Service Management Platform*, page 113. Information Science Reference.

Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I., Montero, R., Wolfsthal, Y., Elmroth, E., Cáceres, J., et al. (2009). The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4–1.

Rodero-Merino, L., Vaquero, L., Gil, V., Galán, F., Fontán, J., Montero, R., and Llorente, I. (2010). From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240.