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, Liège, Belgium*

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 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 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. However, current offerings still present inherent interoperability and portability issues. Interoperability enables different services deployed on different cloud infrastructures to exchange through common interfaces, without the need to provide a specific adapter for each separate API. Portability introduces common formats by which services are described and deployed independently from the provider.

To tackle these issues, we operated at the IaaS Service Manager level and we selected Open Cloud Computing Interface (OCCI)[1] as the cloud management interface. Reservoir (Rochwerger et al., 2009) proposes a well defined architecture as well as an open source implementation. Within this architecture, services are comprised as sets of inter-related virtual machines along with their attached storage, networks and configuration settings. Service managers help users switch from individual VM management to whole services management, thus focusing more on their business goals. On the other hand, OCCI provides common mechanisms for services description, discovery and management. The OCCI-based service definition is processed by the service manager which then handles the deployment, provisioning and SLA enforcement of the service.

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 to seamlessly manage the services deployed within standards-based cloud offerings.

The paper is organized in three parts. To solve the above-mentioned issues, Section 2 introduces our approach based on cloud computing open standards and highlights the design and implementation of our solution. Section 3 proposes future research directions to tackle the remaining issues. Section 4 describes related works and how our solution goes beyond the state of the art.

[1]http://occi-wg.org

2 OCCI SERVICE MANAGER

2.1 OCCI

OCCI is an initiative from the Open Grid Forum (OGF)[2] to provide a standard RESTful Protocol and API for the management of cloud resources. OpenNebula and OpenStack already provide OCCI compliant interfaces (Edmonds et al., 2012).

[2]http://www.ogf.org/

Ghrab A., Skhiri S., Koener H. and Leduc G..
Towards a Standards-based Cloud Service Manager.
DOI: 10.5220/0004372201270130
In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science* (CLOSER-2013), pages 127-130
ISBN: 978-989-8565-52-5

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.

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 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 and Delete operations.

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.

2.2 OCCI Extension

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. Here we suggest an extension to the core specification of OCCI and we built our service manager atop. The extension introduces the concepts related to cloud IaaS service management such as Service, Role, Group and dependency (see Fig. 1):

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 2 depicts such scenario.

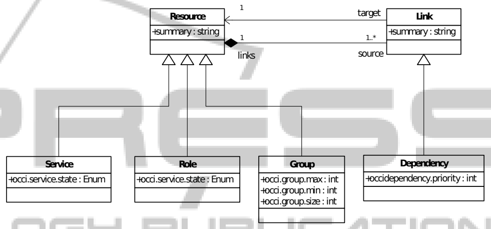

Figure 1: Service Manager Class Diagram.

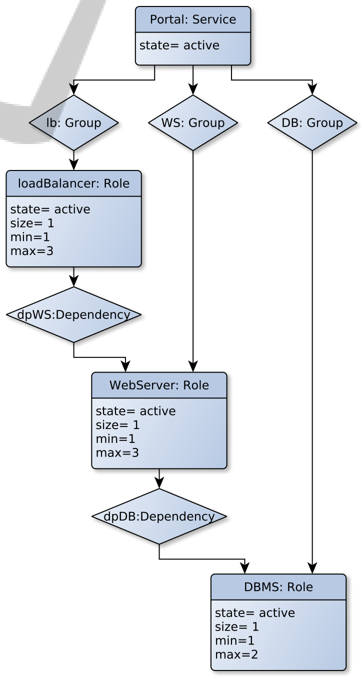

Figure 2: Service Deployment Example.

2.3 Implementation

We have implemented OCCI Service Manager[3] as a proof-of-concept for our extension to the OCCI model. The framework allows the user to define the

[3]https://github.com/KoenerHerve/Service-Manager

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[4] 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 3.

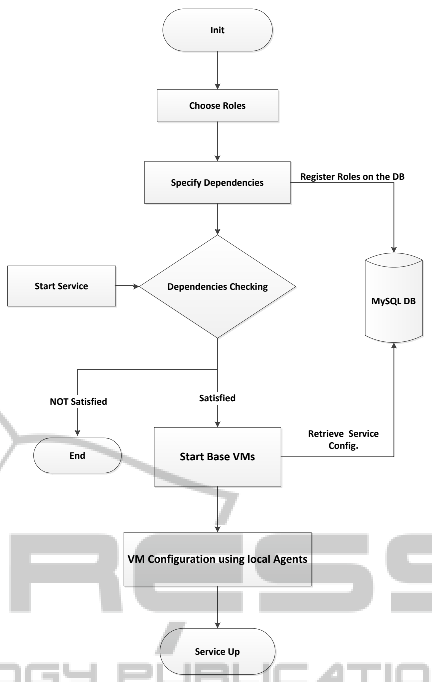

Figure 3: Service provisioning process.

3 FUTURE WORKS

A service manager is assigned two main tasks. First, resources provisioning and deployment, which our tool already supports. Second, monitoring and SLA compliance enforcement. Currently, we do not include monitoring modules. We envision integrating Ganglia[5] as our distributed monitoring system. Scheduling helps optimize resource utilization. Haizea[6] 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.

4 RELATED WORKS

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[7] and Scalr[8]. Deltacloud[9] is providing a single entry point to developers, while maintaining a driver per API backend.

Both approaches are interesting and widely adopted. 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, we chosen to rely on open standards. Here the API maintenance is handled by the cloud provider that comply with the standards.

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

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

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

[7]http://www.rightscale.com

[8]http://scalr.net/

[9]http://deltacloud.apache.org/

Claudia is a service manager supporting service management on 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. We consider that since OCCI is designed for service management, it fits natively better as a service descriptor than OVF.

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)[10] standard, a DMTF model to describe entities and their attributes. The IaaS Aggregator is a cloud management console supporting resources lifecycle management. 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. Furthermore, network topology descriptions are not very well supported by CIM (Ghijsen et al., 2012).

5 CONCLUSIONS

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.

ACKNOWLEDGEMENTS

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

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

REFERENCES

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.

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.

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

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.