PaaS ELEMENTS FOR HOSTING SERVICE-BASED APPLICATIONS*

Sami Yangui[1,2] and Samir Tata[1]
[1]*Institut Telecom, Telecom SudParis, UMR CNRS Samovar, Evry, France*
[2]*Faculty of Science of Tunis, University Tunis El Manar, Tunis, Tunisia*

Keywords: PaaS, Service Deployment, Application Deployment.

Abstract: Cloud Computing is a new supplement, consumption, and delivery model for IT services based on Internet protocols. It typically involves provisioning of dynamically scalable and often virtualized resources. In this environment, there are several issues related to the inadequacies of hosting platforms and mechanisms to ensure the smooth developing and running of service-based applications (communication protocols, ESB, Service containers, etc.). In this paper, we present a new comer of platform as a service (PaaS) based on our already developed service micro-containers to avoid compatibility and portability constraints imposed by classical Cloud platforms. Several platform use cases are discussed.

1 INTRODUCTION

Cloud Computing is a specialized distributed computing paradigm (Foster et al., 2008). It differs from traditional ones on the fact (1) it is massively scalable, (2) it can be encapsulated as an abstract entity that delivers different levels of services to customers outside the Cloud, (3) it is driven by economies of scale and (4) can be dynamically conFigured (via virtualization or other approaches) and delivered on demand.

As a part of our work, we focus on deployment and execution application aspects in the Cloud especially for service-based applications. Taxonomy of Cloud Computing systems shows that all the existing systems which supports service-based application deployment and execution are limited to a programming framework which makes the use of those Clouds difficult, since Cloud clients need to use the related programming language before using the Cloud (Rimal et al., 2009). For example, Amazon imposes Amazon Machine Image (AMI) and Amazon MapReduce framework (Amazon, 2011), Force.com imposes Apex language for database service (Force, 2011), Azure imposes Microsoft.Net (Microsoft, 2011), Google App Engine imposes MapReduce programming framework (Dean and Ghemawat, 2004) (Engine, 2011), and so on.

Similarly, the same type of constraints are imposed on developers to deploy service-based applications in the Cloud raises often portability and compatibility issues. Concretely, Cloud providers tries to diversify to the maximum the offered API to access to their Cloud services. We can cite among other the following APIs: Amazon API (Varia, 2011), GoGrids API (GoGrid, 2011), Suns Cloud API (Sun, 2011) and VMwares vCloud. However, no Cloud provider can provide the all existing API for access and deployment in Cloud platforms and therefore cannot satisfy all the portability and compatibility constraints that can meet developers community in the Cloud. In this paper, we propose a newcomer of platform as a service (PaaS) which allows developers to deploy, host and run service-based applications regardless of their implementations and APIs. The paper is organised as follows : Section 2 presents a state of the art of classical Cloud platforms. Our findings are discussed in section 3. Finally, we conclude our paper and present our future work in Section 4.

2 STATE OF THE ART

In this section, we present a state of the art of different classical PaaS architectures. We are interested essentialy on PaaS components and containers which inter-

*The work presented in this paper was partially supported by the French FUI CompatibleOne project. http://compatibleone.org/bin/view/Main/

Yangui S. and Tata S..
PaaS ELEMENTS FOR HOSTING SERVICE-BASED APPLICATIONS.
DOI: 10.5220/0003932204760479
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 476-479
ISBN: 978-989-8565-05-1

venes in service deployment and execution processes to study platforms behavior on deployment taks and highlight API limitations support of these platforms.

Google offers Google App Engine (GAE) (Engine, 2011) is one of the platforms which offer devlopement and deployment features for applications coupled to the GAME API (Zahariev, 2009). Among the interesting features that provide the platform, there is a possibility to connect to the internal SI securely. There is also Heroku PaaS which is optimized for Web applications developement using Ruby and Rack (Heroku, 2011). The functioning of Heroku is quite simple: The infrastructure used is Amazon Web Services (AWS) through Internet while the deployment and the management of applications is done using Git. Microsoft is also a main actor in Cloud PaaS solutions. Microsoft Azure offers typical data storage (SQL Azure Storage), running applications on Web servers (IIS Web Role), an equivalent of Windows services (Work Role), bus systems and data management access (Microsoft, 2011).

We can also quote open source PaaS like Cloud Foundry (Xebia, 2011), CloudBees (CloudBees, 2011) or OpenShift (OpenShift, 2011)

A comprehensive study of these platforms allowed us to highlight limitations and constraints imposed by classical PaaS to developer community. For example, in CloudBees platform, *RUN@CLOUD* service supports only languages that are based on JVM like Scala, Clojure or Jruby, even if that there is one platform user who define a Grails CloudBees plugin (Plugin, 2011) and he still to maintain it regularly. Similarly, the specific JVM and APIs offered by GAE are not 100% standard and requires adapting applications to deploy to take advantage of the power of the platform even if the API is high-level and a number of common frameworks and JVM-based languages are supported. Another limitation observed concerns Microsoft Azure which supports only .NET applications. In addition to that, we note the restrictions on platforms communication protocols and on bindings implementations of deployed services. For example, *Router* component from Cloud Foundry and *Reverse proxy* component from Heroku allows only HTTP messages ingoing and outgoing platforms.

Our objective is to propose a new comer of PaaS which is independent of any API Cloud platform. To do this, we have to resume our previous work on micro-service containers. Service micro-containers are scalable and come adress limitations of classical service containers in Cloud environments (Yangui et al., 2011).

3 CLOUDSERV: A PaaS FOR SERVICE-BASED APPLICATIONS

In this section, we firstly explain briefly our previous works on scalable service micro-containers. Then, we present our PaaS called CloudServ. We describe several deployment scenarios using CloudServ. We first treat the case of simple construction and deployment of an elementary service before moving to the case of deploying a service-based applications.

3.1 Previous Work

As a part of our work, we defined a new approach for service-based application deployment and execution on the Cloud. We designed a newcomer of service containers which can be scalable (Mohamed et al., 2011). We got the idea to create a service micro-container that is able to contain not more than one service. This micro-container provides the minimal functionalities to manage the life cycle of the deployed service. With this idea we have shown that we used the minimal resources to encourage the pay as-you-go model of Cloud Computing (Grossman, 2009) and we could enforce the elasticity of Cloud because we use just the resources needed. We have also conducted conclusive experiments against classical service containers (like Apache Axis2) to validate the reliability and scalability on micro-containers (Yangui et al., 2011).

Since we consider several types of services (languages, bindings, etc.), we are able to generate the correspondent micro-container for each service to be deployed. Micro-containers are built dynamically from the deployment framework that we have realized as part of our work (Yangui et al., 2011).

As shown in (Figure 1) , each service micro-container consists of three modules: (1) Communication module to establish communication and to support connection protocols, (2)Processing module to process ingoing and outgoing data into and out of the server (packing and unpacking data) and (3)Service module to store and invoke the requested service.

In the following subsection, we show how micro-containers are generated and deployed in order to deploy elementary services on ClouServ. In subsection 3.3, we show how CloudServ support service-based application deployment from deploying a set of elementary service compositions.

3.2 Elementary Service Deployment

Figure 1 presents our deployment framework. To de-

ploy an application in a micro-container, one must mainly provide for the deployment framework two elements: the application with all its components (code, resources, etc.) and a deployment descriptor that specifies the container options to run the application (Figure 2, action 1). Processing module analyzes then the sources, detects the service bindings types and associate to the service sources a communication module implementing these bindings and a processing module to run the service. The resulting code represents the generated micro-container code. It is composed only of the necessary modules for the deployed service, no more, no less (Yangui et al., 2011). The framework generates then a micro-container which hosts the service and implements its bindings regardless of its communication protocol support as long as they are included in Generic communication package (Figure 2, action 2).

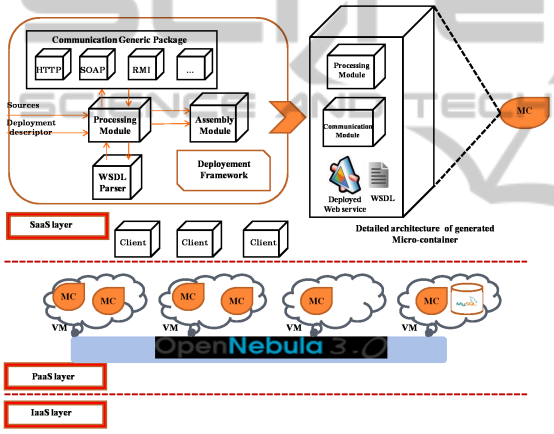

Figure 1: Global architecture of the performed system.

Henceforth we can proceed to the deployment task. We deploy the obtained micro-container in one of the created platform VMs (Figure 2, action 3). After that, we can invoke it using a client from the SaaS layer (Figure 2, action 4) or even from another deployed micro-container on CloudServ (Figure2, action 5).

Once deployed, the service is completely autonomous. If it is invoked via its micro-container, it runs locally and returns the execution result to the requester before processing any other requests. The Cloud platform we used is very simple. It consists of a set of VMs created using OpenNebula from IaaS layer resources as described in Figure 2.

3.3 Application Deployment

For service-based applications deployment on CloudServ, we proceed by decomposing the application into

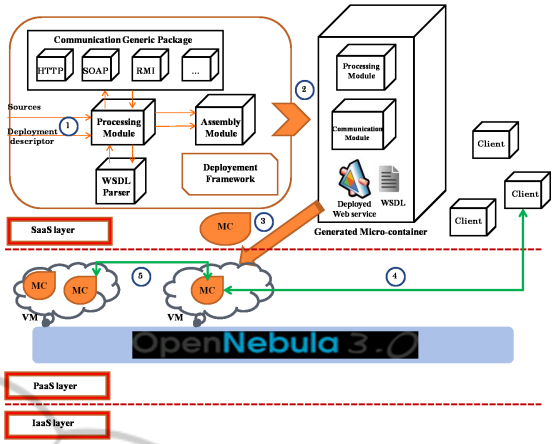

Figure 2: Deployment steps of an elementary service on the platform.

a set of elementary services. More explicitly, for each application deployment request, we try to divide it into multiple elementary services deployment requests. We consider two specification of service composition: Service Component Architecture (SCA) (SCA, 2011) and Business Process Execution Language (BPEL) (Juric, 2006). BPEL consumes only SOAP-based service orchestrations (Weaver, 2005). Besides, SCA supports heterogeneous protocols recovering several binding component implementation. In this paper, we treat the case of SCA. BPEL treatment can be easily deduced from our proposal.

Defining an orchestration between such components, which have different implementations and various bindings types, makes deployment task very consistent and complicated in a Cloud platform due to the difficulty to satisfy all the compatibility and portability constraints on the platforms mentioned later. Overall, the developer has to provide to the deployment framework the composite which describes an orchestration scenario. It should be noted that the developer can provide only the services that are not already deployed on the platform; it can make a simple reference to the elementary deployed (processed in subsection 3.2) services in the composite.

We performed a new component called SCA parser and we integrated it into the deployment framework (Figure 3). This component is responsible for parsing the composite provided by developer with the service-based application at the submission of the deployment request (Figure 3, action 1). Deployment framework generates correspondent micro-containers taking into account the orchestration scenario and service binding types interacting between these components (Figure 3, action 2). Thus, once the micro-containers deployed on the VMs platform, they can

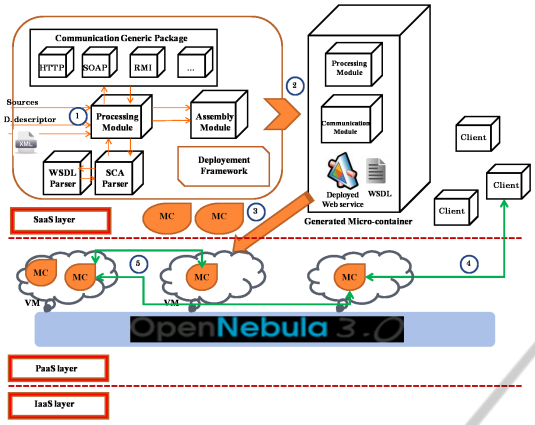

Figure 3: Deployment steps for service-based application.

participate in a composition scenario to achieve functionality of a service-based application (Figure 3, actions 4 and 5). Specifically, each micro-container retrieves a set of inputs needed to the execution of its service from the preceding in the global execution process of a service-based application, runs locally, and sent to its successor a set of references for its execution.

4 CONCLUSIONS

In this paper, we highlighted portability and compatibility constraints imposed by classical Cloud providers and platforms making development and deployment tasks difficult and delicate for users. We have proposed a new prototype of platform as a service (PaaS) that tackles these drawbacks called CloudServ. CloudServ is based on reuse of the scalable service micro-containers introduced as a part of our previous work. Only necessary resources to implement service binding types, such as communication protocols, are selected from the deployment framework of the system and encapsulated in the generated micro-container to host the deployed service. We also presented and described several CloudServe use cases for deployment of service-based application. In the near future, we plan to deepen our experiments of CloudServ versus several Cloud platforms with various scenarios to validate it.

REFERENCES

Amazon (2011). Amazon web services. http://aws.amazon.com/fr/.

CloudBees (2011). Cloudbees. http://www.cloudbees.com/.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation*, OSDI'04, San Francisco, USA.

Engine, G. A. (2011). Google app engine. http://code.google.com/appengine.

Force, D. (2011). Salesforce.com and force.com developer resources. http://developer.force.com/.

Foster, I., ans I. Raicuand, Y. Z., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *The IEEE Grid Computing Environments*, GCE'08, Austin, USA.

GoGrid (2011). Gogrid. http://www.gogrid.com.

Grossman, R. (2009). The case for cloud computing. *IT Professional*, 11 Issue:2:23–27.

Heroku (2011). Heroku platform. http://www.heroku.com/.

Juric, M. B. (2006). *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition.* ISBN:1904811817.

Microsoft (2011). Microsoft azure. http://www.microsoft.com/azure/default.mspx.

Mohamed, M., Yangui, S., Moalla, S., and Tata, S. (2011). Service micro-container for service-based applications in cloud environments. In *IEEE WETICE*, Paris, France.

OpenShift (2011). Red hat openshift. https://openshift.redhat.com/app/.

Plugin, G. C. (2011). Grails cloudbees plugin. http://grails.org/plugin/cloud-bees.

Rimal, B., E.Choi, and Lumb, I. (2009). A taxonomy and survey of cloud computing systems. In *In the fifth International Joint Conference on INC, IMS and IDC*.

SCA (2011). Sca specifications. http://www.ibm.com/developerworks/library/specification/ws-sca/.

Sun (2011). The sun cloud api. http://kenai.com/projects/sunCloudapis.

Varia, J. (2011). Amazon white paper on cloud architectures. http://aws.typepad.com/aws/2008/07/white-paper-on.html.

Weaver, R. (2005). The business value of the service component architecture (sca) and service data objects (sdo). In *IBM whitepape*.

Xebia (2011). Xebialabs deployement lifecycle management. http://blog.xebia.fr/2011/04/13/lancement-du- projet-platform- as-a- service-cloud- foundry-de-spring-source/.

Yangui, S., Mohamed, M., Tata, S., and Moalla, S. (2011). Scalable service containers. In *IEEE International Conference on Cloud Computing Technology and Science, (IEEE CloudCom 2011)*, Athenes, Greece.

Zahariev, A. (2009). Google app engine. *TKK T-110.5190 Seminar on Internetworking*.