PaaS Federation Analysis for Seamless Creation and Migration of Cloud Applications

Daniel A. Rodríguez Silva[1], Lilian Adkinson-Orellana[1], Dolores M. Núñez-Taboada[1] and F. Javier González-Castaño[1]

[1]*Gradiant, Network and Applications Area, Ed. Citexvi, 36310 Campus Vigo, Spain*

{darguez,ladkinson,dnunez,javier}@gradiant.org

Keywords: Cloud federation, Platform as a Service, modular programming, provider lock-in, interoperability.

Abstract: A major concern regarding the development of cloud applications is provider lock-in. There exist many platforms with profound technological differences, so it is not possible to port code between them. There are no standard solutions to this problem, with the exception of some attempts of achieving interoperability at the infrastructure level. This paper presents an analysis focused on the federation at PaaS level. Furthermore, the proposed schemes are intended to simplify the design of cloud applications by means of a declarative programming based on a set of modules available in all supported platforms. Hence, designed applications can migrate between platforms easily, enabling the creation of new fault-tolerant architectures.

1 INTRODUCTION

The Platform as a Service (PaaS) paradigm offers a simple way of building cloud applications, hiding the complexity of the underlying infrastructure to developers (Lawton, 2008). Nevertheless, there exist several PaaS from different providers, each with its own advantages and disadvantages. For this reason, in some cases it is difficult to choose the most appropriate. Cloud platforms usually provide a friendly development environment to help in the design and deployment of applications in the cloud. However, when a developer selects a platform, its applications usually get tied to the underlying technologies, hindering the migration to other PaaS (Louridas, 2010).

Some popular platforms are Google App Engine, Microsoft Azure, Heroku, Force.com or GigaSpaces XAP. These few examples are highly heterogeneous in terms of supported languages (Python, Java, .NET, Ruby...), environments and programming tools (NetBeans IDE, Eclipse IDE, Visual Studio, specific web portals...). This illustrates the strong dependence inherent to the adoption of a particular solution. Moreover, the complexity of transferring large amounts of data between cloud providers makes migration between platforms particularly difficult. Therefore, users have no option but accepting unexpected changes on the conditions of contracted services (Bradshaw, Millard and Walden, 2010).

For this reason, an abstraction layer over different PaaS, which will allow easy development and seamless deployment of applications in any of them, is of great interest. The concept of PaaS federation minimizes the problems aforementioned, as an intermediate layer that guarantees interoperability between PaaS providers, avoiding the dependence on specific technologies. In this paper we introduce this concept. Section 2 reviews related work. Section 3 analyzes the features of the federated PaaS and presents three different approaches for its architecture, with their advantages and disadvantages. Finally, section 4 concludes the paper.

2 RELATED WORK

Even though the federation of cloud resources is an extended concept at the infrastructure level (Buyya, Ranjan and Calheiros, 2010) and (Rochwerger, Breitgand, Levy, Galis and Nagin, 2009), it does not have a clear equivalent at the platform level.

CumuLogic (CumuLogic, 2010) is a service that allows to create private and customized PaaS on

public (Amazon EC2) and private clouds (Cloud.com, Eucalyptus, VMWare), for building Java applications. It also offers a set of integrated and tested infrastructures, selectable from a catalogue and ready to use as a base for the designed platform. However, the migration of developed applications to other existing public or private PaaS is complex.

Cloud Foundry (Cloud Foundry, n.d.) is an open source platform deployable on several public (Amazon Web Services, RackSpace, GoGrid…) or private clouds. This VMWare PaaS is based on the idea of providing an open platform that facilitates to change the infrastructure provider transparently to its users, although it does not operate at Platform level. The same idea appears in Red Hat OpenShift (Red Hat OpenShift, n.d.), a free platform that allows developing applications in Java, Python, PHP and Ruby. The platform is based on the Deltacloud interoperability standard (Deltacloud, 2011), permitting to run applications on any infrastructure of the Red Hat Certified Public Cloud Provider.

mOSAIC (Di Martino, Petcu, Cossu, Goncalves, and Máhr, 2011) is a FP7 project that seeks an open-source platform in which applications will negotiate cloud services as requested by their users. The platform will implement a multi-agent brokering mechanism that will search for services matching application requests. Cloud application developers and maintainers will be able to rely on runtime mechanisms for the procurement of cloud services, while end-user applications will find the best-fitting cloud services to their actual needs and efficiently outsource computations. Nevertheless, again, this solution focuses mainly on the infrastructure level and does not consider specifically the PaaS level.

In (Paraiso, Haderer, Merle, Rouvoy and Seinturier, 2012) a federated multi-cloud PaaS infrastructure is presented. This configurable infrastructure is an open service model to design and implement both a multi-cloud PaaS layer and the SaaS applications running on top of it. However, it considers some infrastructure services for managing both the multi-cloud PaaS and the SaaS applications so it also relies on the IaaS layer. Moreover runtime components are implemented in Java, preventing the platform to assist PaaS that do not support this programming language.

In (Gonçalves, Cunha, Neves, Sousa and Barraca, 2012) a cloud service broker is presented. It allows developers to manage their applications through a web portal or a command line that make use of a PaaS and an IaaS manager. These managers support multi-provider and multi-cloud environments using one only API. This platform helps developers to decide the best provider to fulfil their requisites. However, the main drawback is that the framework does not make the selection of implementation technologies transparent, limiting the PaaS where the applications can be deployed.

3 PAAS FEDERATION ANALYSIS

3.1 Federated PaaS requirements

A federated PaaS is a high level platform that supports a set of traditional PaaS to provide an abstraction layer simplifying the creation and deployment of cloud applications. In order to build a federated PaaS it is important to consider some requisites and functionalities.

3.1.1 Ease of application design

The development of applications in the federated PaaS is reduced to a simple process of selecting and configuring a set of predefined modules. Each module provides a different functionality available in all the underlying platforms. Modules should be created beforehand and mapped to the corresponding implementations in each supported platform. In this way, application development becomes a simple choice of the appropriate modules to obtain the desired functionality. Following this approach of modular declarative programming it is possible to develop applications at lower effort. Figure 1 illustrates the concept proposed. Modules are represented by coloured boxes connected through the different cloud layers. Each module can work autonomously and has a well-defined interface (SOAP, REST…) to facilitate interoperability.

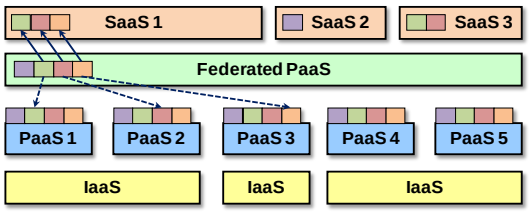

Figure 1: General architecture of the proposed system.

The federated PaaS will also provide mechanisms to add modules to its repository according to the needs of the SaaS developers. However, the new modules will need to be mapped into the underlying platforms to offer a full functionality.

Moreover, in order to maximize flexibility for developers and administrators a web portal will allow the creation and management of the applications from anywhere.

3.1.2 Monitoring

The federated PaaS will include a monitoring module responsible of verifying that the quality parameters of interest stay within the established thresholds.

The selected measurable parameters are standard ones: input and output bandwidth (GB), consumed RAM memory (MB), consumed CPU time (CPU hours) and stored data (GB per month). This information will allow billing based on average resource consumption during a time window, besides the verification of SLA (Service Level Agreement) fulfilment. Furthermore, it is desirable to classify the deployed applications and predict their behaviour by analysing the parameters collected by the monitoring system. This feature enables the integration with an intelligent prediction system to optimize the selection of the most suitable PaaS for each scenario in near real-time.

3.1.3 Efficient resource provisioning

Regarding resource provisioning, the main goal is the creation of an intelligent delivery system capable of determining the most appropriate PaaS fitting application requirements. By default, the federated PaaS will deploy an application in the cheapest platform that fulfils the established SLA. This offers the possibility to migrate the application to another PaaS based on its resource demand or possible changes in SLAs. This functionality will depend on data retrieved by the monitoring module.

3.1.4 Smart deployment

Developers can choose the PaaS to deploy applications manually. The federated PaaS hides the complexity of dealing with underlying PaaS. However, the federated PaaS can also determine the most suitable PaaS for each application considering SLA constraints. This process, which is transparent to developers, will minimize costs by migrating applications to alternative platforms in which their performance improves.

In addition, applications developed on the federated PaaS are intrinsically fault-tolerant: in case of platform failure, applications will automatically migrate to another federated platform satisfying their requirements.

Another interesting feature is the possibility for an application of using modules from different PaaS in order to obtain the best price or performance combining all of them.

3.1.5 Security and privacy

Security is one of the most important concerns regarding adoption of the cloud computing paradigm. In the context of PaaS federation there are two main security considerations. On the one hand, developers should have mechanisms to build secure applications over the federated PaaS. For this reason, a subset of security modules should be available for application designers: a data encryption module, a data integrity module (hash), an encrypted storage module, etc. On the other hand the federated PaaS should provide secure communication with the underlying PaaS, an adequate authentication mechanism, and protection for sensitive information, among other.

3.2 Federated PaaS approaches

The first proposed approach to build a federated PaaS is shown in Figure 2. In the design phase developers can create an application by choosing the desired components from a catalogue and defining the way they are connected. The result is a configuration file that describes how the application must be built. This file is processed by a component (PaaS selector), which is in charge of choosing the most appropriate PaaS attending to SLA constraints and the current status of the underlying PaaS.

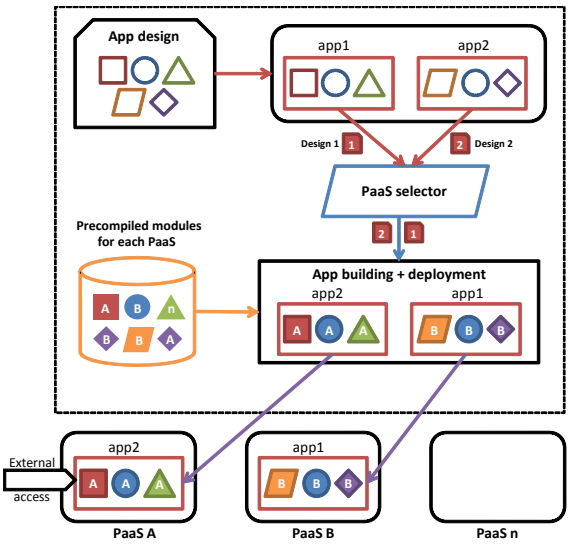

Figure 2: Model A.

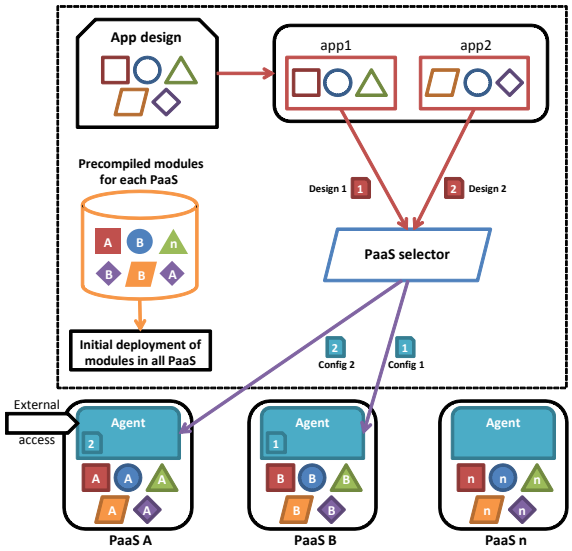

Figure 3: Model B.

Once the target PaaS is chosen by the PaaS selector, the builder component creates the application combining the appropriate modules from the repository and deploying them on the selected platform. Hence, each application created in the federated PaaS will have a unique correspondence with an application deployed on the target PaaS. Besides, all modules belonging to an application will always be deployed on the same platform (there is a unique deployment per application of all its modules).

The main benefit of this model is that it offers the possibility of easily exploiting the monitoring and accounting capabilities of each underlying PaaS. As all the modules of an application are deployed on the same platform, it is straightforward to obtain these values. However, this design has the drawback that, as each deployment on a PaaS needs to be registered, it is necessary to automatize the registration of applications in each underlying platform, which is not trivial.

The second approach (model B) is shown in Figure 3. In this case, in an initial stage, all the predefined modules are deployed in each supported PaaS. Therefore, when a new application is created there is no real deployment, but only an instantiation and configuration of the corresponding modules. To support this scheme, there is an agent in each PaaS in charge of orchestrating the modules according to the configuration file generated in the design phase. This agent receives the specific rules from the PaaS selector to assemble the modules dynamically and acts as the entry point for external access.

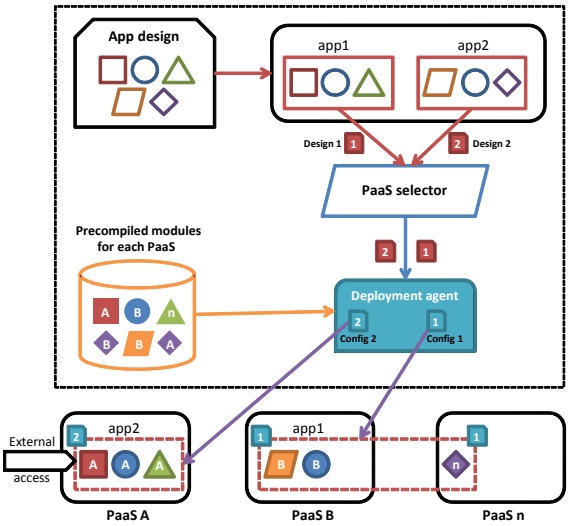

Figure 4: Model C.

In this scenario, as mentioned before, each underlying PaaS registers and deploys all the predefined modules in advance (as if there where individual applications), so that all the designed applications of all users share them. The main advantage of this approach is that deployment time is almost null. In addition, it is possible to combine modules deployed in different platforms to compose a federated PaaS application, which allows developers to benefit from the most suitable option in each case.

The main disadvantage of model B is that, unlike the previous approach, it is necessary to develop a specific federated PaaS monitoring system, because it is required to evaluate and register the resources consumed by each application in each shared module.

The third approach is a hybrid solution combining some features from model A and model B. As shown in Figure 4, the configuration file generated in the design phase is processed by a deployment agent in charge of deploying all the needed modules in the corresponding underlying PaaS. In this case, each module is configured before being deployed following the federated PaaS application rules. Each module will have a logic inside to know how to interact with other modules depending on its configuration.

Thus, the modules of each application in the federated PaaS are deployed as autonomous applications in underlying PaaS layers with appropriate configurations. Modules are not shared in this case. Therefore, several modules of the same type can be deployed in the same PaaS, to be used

by different applications. Additionally, the same application can use several modules from different PaaS. With this model, the federated PaaS can exploit the monitoring capabilities of each PaaS. However, since the modules are deployed on-demand, it is necessary to register and deploy them programmatically.

3.3 Discussion

The three models presented have different characteristics depending on how the federated PaaS interact with the different underlying PaaS. Table 1 compares four important features to be considered, because there is not direct control over public PaaS supported by the federated PaaS.

Table 1: Analysed models comparison.

Feature	Model A	Model B	Model C
Need of dynamical application registration	Yes	No	Yes
Monitoring capabilities exploited	Yes	No	Yes
Applications can combine modules from different PaaS	No	Yes	Yes
Instant deployment of applications	No	Yes	No

Public PaaS as Windows Azure or Google App Engine usually provide an API to access specific functionalities programmatically. However, in some cases the registration of new applications must be manual, via a web interface. For that reason model B is the only valid option when dynamic registration is not available. On the other hand, public PaaS provide mechanisms to let users know the resources that their applications consume in order to verify their billing. This feature can be exploited in models A and C, but model B requires implementing a specific monitoring system. Regarding combination of modules deployed in different PaaS to build an application, model A is the only one that cannot support this, because applications are specifically built for the PaaS where they are going to be deployed. Combining modules from different PaaS in one application brings flexibility, for example when some module is only available for one PaaS. However this can be dangerous in the sense the final application cannot always migrate to another PaaS. Finally, model B is the fastest option to deploy final applications as its modules are only instantiated and configured, so that it is not necessary to deploy them individually each time.

To sum up, the option that has less dependency with regard to the underlying PaaS is model B, despite a monitoring system has to be implemented. If all supported PaaS have a complete API to manage applications, model C is the most flexible option.

4 CONCLUSIONS

Due to the variety of current PaaS solutions, it is not easy to unify criteria for the development of applications, so further work is needed to map the functionalities of the provided modules –at federated PaaS level– for each supported platform. The current state of the art indicates that this problem is still not solved at all, since there are not any attempts of unification exclusively at the PaaS level. Application development independently of the underlying PaaS technologies mitigates the problem of provider lock-in, which hinders the migration to other PaaS and, as a consequence, provides fault tolerance across different vendors. The three architectures analysed give an idea of the difficulties involved in federation at PaaS level. Developers must pay the price of lower freedom for creating applications, since they will have to rely on a set of predefined modules. However, this problem can be mitigated by adding modules on demand, as our platform is designed to be easily extended.

ACKNOWLEDGEMENTS

This research has been supported by the CloudMeUp grant (IDI-20101357), funded by CDTI, Spain.

REFERENCES

Bradshaw, S., Millard, C. and Walden, I. (2010). Contracts for clouds: A comparative analysis of terms and conditions for cloud computing services. In *Queen Mary School of Law Legal Studies Research (Paper No. 63/201), London.*

Buyya, R., Ranjan, R. and Calheiros, R. N. (2010). InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Algorithms and architectures for parallel*

processing, Lecture Notes in Computer Science, vol. 6081/2010, pp.13-31.

Cloud Foundry, (n.d.). *VMWare Cloud Foundry website*. Retrieved January 21, 2013, from http://www.cloudfoundry.com/

CumuLogic (2010). *CumuLogic website*. Retrieved January 21, 2013, from http://www.cumulogic.com/

Deltacloud (2011). *Deltacloud standard website*. Retrieved January 21, 2013, from http://deltacloud.apache.org/

Di Martino, B., Petcu, D., Cossu, R.. Goncalves, P., Máhr, T. and Loichate, M. (2011). Building a Mosaic of Clouds. In *Euro-Par 2010 Parallel Processing Workshops, Lecture Notes in Computer Science, vol. 6586/2011, pp.571-578.*

Gonçalves, C., Cunha, D., Neves, P., Sousa, P. and Barraca, J. (2012). Towards a Cloud Service Broker for the Meta-Cloud. In *CRC 2012, Construction Research Congress*, West Lafayette, IN, U.S.A.

Lawton, G. (2008). Developing Software Online with Platform-as-a-Service Technology. *Computer, vol. 41, no. 6, pp. 13-15*.

Louridas, P. (2010). Up in the Air: Moving Your Applications to the Cloud. *Software, IEEE, p. 6-11*.

Paraiso, F., Haderer N., Merle P., Rouvoy R. and Seinturier L., (2012). A Federated Multi-Cloud PaaS Infrastructure. In *CLOUD 2012, 5th IEEE International Conference on Cloud Computing*, Honolulu, Hawaii, EEUU.

Red Hat OpenShift (n.d.). *Red Hat OpenShift website*. Retrieved January 21, 2013, from https://openshift.redhat.com/app/

Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente I. et al. (2009). The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development, vol. 53, no. 4, pp. 1-11*.