Towards a Cloud Service Broker for the Meta-Cloud

Carlos Gonçalves*, David Cunha†‡, Pedro Neves†, Pedro Sousa‡, João Paulo Barraca*, Diogo Gomes*
*Instituto de Telecomunicações, University of Aveiro, 3810-193 Aveiro, Portugal
{cgoncalves,jpbarraca,dgomes}@av.it.pt
†Portugal Telecom Inovação, SA, 3810-106 Aveiro, Portugal
{david-g-cunha,pedro-m-neves}@ptinovacao.pt
‡Centro ALGORITMI / Department of Informatics, University of Minho, 4710-057 Braga, Portugal
pns@di.uminho.pt

***Abstract*—Cloud Computing provides computing resources, middleware and (web-based) software on an on-demand basis. This model helps customers saving costs and allows access to the latest technology. With the exponential growth of IT companies offering cloud services, deploying applications to the cloud has become a complex task to engage. Almost each and every provider has its own terminology, providers do not share the same (or even similar) API, and costs of operation greatly diverge according to provider, region or availability.**

This paper propounds a Cloud Service Broker (CSB), and describes an early prototype, where users are, intelligently and autonomously, aid to deploy, manage, monitor and migrate their applications in a cloud of clouds. A single API is required to orchestrate the whole process in tandem with two truly decoupled managers: a Platform as a Service Manager (PaaS Manager) and an Infrastructure as Service Manager (Iaas Manager). Users also interact with the CSB through a Web portal and a command-line interface.

***Index Terms*—Cloud Service Broker, IaaS, PaaS, Cloud Interoperability, SOA**

I. Introduction

Cloud Computing is an emerging technology used to deliver on-demand services over the Internet. It is undoubtedly affecting the way business is conducted and is empowering a new generation of products and services. Cloud Computing can be summarized into three keywords: elasticity, on-demand, and (autonomously) fully-managed [1]. These three characteristics massively benefit organizations by reducing both CAPEX and OPEX while enabling them to channel their efforts to the strategic business sector [2].

Business models for cloud computing present the resulting system according to three major layers: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [3]. Several companies such as Amazon, RackSpace, Google, and Microsoft, expose their cloud solutions publicly as business-to-client and business-to-business models. These companies provide on-demand cloud services through a diverse set of business models and prepositions. Availability, continuity, API support, location, charging scheme and price, as well as technical characteristics such as computational capabilities, storage and processing memory, are some of the parameters of the resulting business offers.

Consider the following short list of presently available proprietary and open source PaaS software platformas available for public consumption. Heroku[1] is a well known cloud application provider supporting a variety of runtime languages including the Java, Ruby, Python and Node.js frameworks as add-ons. Microsoft's Windows Azure[2] is a 99% Service-Level Agreement (SLA) cloud platform proportioning users with extended services like Content Delivery Network (CDN), application load balancing, and service bus. On the open source side of PaaS platforms, efflorescent projects has come to light recently with a successful prospective enabling cloud provider enterprises to embrace such projects and contribute back to a better product, as well as enabling users to run their own PaaS instance on a private cloud. Cloud Foundry[3] is a open source platform project initiated by VMWare Inc. Cloud Foundry strives to be multi-language, multi-framework, multi-service, and multi-cloud. It is also in its goals to be horizontally scalable and self healing with no single point of failures, and fail fast, following basic patterns–event-driven, asynchronous, and non-blocking [4]. All these solutions are limited because cross-cloud management functionalities are lacking [5], and they are focused in interoperation, forgetting about many self-healing and business continuity aspects. Cloud consumers are therefore faced with the problem of choosing the best provider, considering the factors identified as critical for each owned system and individual applications.

The framework we propose addresses the aforementioned issue by aiming to develop a Cloud Service Broker (CSB) which eases crucial lifecycle application stages (deployment, management, and monitoring) from developers orchestrating the whole process in tandem with a PaaS Manager and an IaaS Manager. To achieve such goal, the CSB should be aware of user application requirements, be aware of available cloud offers, and transparently deploy to the best rated cloud provider based on given premises, being either and ideally a PaaS or if premises are not fulfilled, falling back to an IaaS solution in which a PaaS framework is setup on demand. Through the proposed CSB, end-user developers will have access to a feature-complete Web portal where they can auto-deploy, manage and monitor their appliances. Ultimately, the CSB allows the integration of intelligent mechanisms towards

[1] http://www.heroku.com/
[2] http://www.windowsazure.com
[3] http://www.cloudfoundry.com/

the live migration of SaaS between PaaS and live migration of PaaS between IaaS, in such a way that it will self heal from a misbehaving provider or in response to localized service demand, or cost cutting by switching-over to a less expensive provider.

The remainder of this paper is structured as follows. Section II briefly overviews the state of the art in the cloud computing area focusing on cloud mediators, decision making helpers and cloud API standards. Section III proposes a novel framework solution for the needs identified, which by taking a bottom-up approach describes two new decoupled products, the PaaS Manager and IaaS Manager. Finally, Section IV concludes the work presented.

II. Related Work

Industry and academic related work have been published throughout the last two years actively. Work includes, but not restricted to, decision making engines for cloud services, unified multi cloud management systems and cloud brokering. The majority are focused on IT infrastructure.

The work in [6] discusses a methodology for a multi-criteria cloud service selection. Cloudle [7], another multi-criteria engine, uses cloud ontologies for IaaS and PaaS, amongst others "as a Service" cloud offerings. On multi cloud manageability [8] proposes a platform that mediates between users and IaaS resources and hence reducing workload for IT operators by only requiring a single sign-on in one unified portal. A proven implementation of this kind of cloud management concept is Red Hat's sponsored Aeolus project[4].

An interesting reading is [9]. It identifies and compares common use cases on different IaaS and PaaS cloud platforms. Interfaces, documentation, ease of use and time taken to learn, and number of steps required to accomplish a task are measured.

Because hardly any PaaS offering will ever satisfy every application or system requirement, a new approach needs to be considered to fill in the gap. An approach which really integrates different cloud architectures, providing enhanced management capabilities. A first approach is to interoperate services across different providers creating a whole new and tailor-oriented PaaS per tenant. However, in order to consider a broader approach, we can not safeguard an absolute coverage of all functional prerequisites. As such an extended approach has to be contemplated to tackle the downside of the former strategy. The proposed approach presented in this paper requires the use of less abstract interfaces to the cloud, namely through the use of IaaS solutions, and perform some automatic and transparent work from the point of view of the cloud consumers. The objective is to meet the requirements specified by users from the set of available cloud providers, which could not be accomplished by existing single PaaS offerings.

Also considerable efforts on the cloud API standardization have been conducted by industry organizations. They are committed to unify incompatible APIs from various cloud providers as a single one. Examples of well-defined IaaS APIs are Distributed Management Task Force (DMTF)'s Cloud Infrastructure Management Interface (CIMI)[5] and Open Cloud Computing Interface (OCCI)[6] from Open Grid Forum (OGF). These two standards are already being used in a considerable number of projects such as on OpenStack, CloudStack, and OpenNebula. The freshly unveiled Cloud Application Management for Platforms (CAMP)[7], a joint effort by big companies including Oracle, Red Hat, Rackspace and others, strives to bridge the same gap as CIMI and OCCI do though on the PaaS layer of the cloud stack. PaaS management API's CAMP goal is to provide a common ground across multi-cloud solutions. Consumers interact in a REST-based approach, exchanging JSON encoded data for the model resources. Interoperability between platform clouds is made easier.

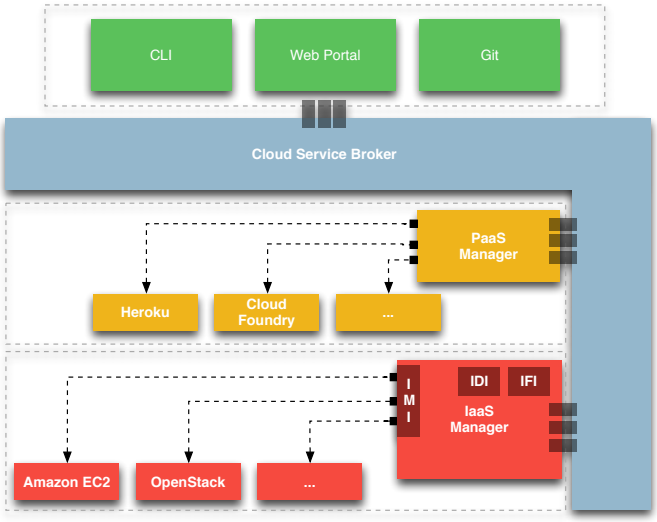

Fig. 1: Architecture overview

III. Proposed Architecture

The hereby presented CSB framework, and depicted in Figure 1, consists on four layers. The PaaS and IaaS managers support multi-provider and multi-cloud environments using each a single API, orchestrated by the intelligent CSB. Users have access to a Web portal and command-line interfaces where they can perform arbitrated operations on resources.

Combining all service oriented components of multiple cloud systems into one single piece of software, provides an efficient and practical platform. This approach is suitable not only for helping users decide which PaaS provider better fits current or future requirements in terms of deployment, execution and monitoring, but also for the case that no single provider fulfills these requirements. Intelligent handling of application specification enables fallback to different PaaS providers or smooth deployment to be built directly in to a IaaS cloud solution.

[4]http://aeolusproject.org

[5]http://dmtf.org/standards/cloud
[6]http://occi-wg.org
[7]http://cloudspecs.org/CAMP/

A. Platform as a Service Manager

The Platform as a Service Manager (PaaSManager), detailed in this section, is a framework which aggregates several PaaS public offerings based on shared similarities. The PaaSManager is intended to provide fundamental features for developers, such as, create, manage, monitor and acquiring information regarding applications and databases. Furthermore, the portability of applications between vendors is a foremost purpose of this architecture. For the development process of such approach, some PaaS offerings were selected to belong to an interoperable ecosystem.

Besides well-known vendors like Amazon, Google and Windows, there are several platforms which offer attractive solutions for developers. The selected platforms, CloudBees, CloudFoundry, IronFoundry and Heroku, were broadly studied as well as their native APIs and management processes. As result, 20 operations were specified based on similarities shared by the platforms from the ecosystem. Some additional developments had to be performed in order to provide a complete transparency for developers, like supporting Git as deployment tool for every PaaS. Figure 2 resumes some of the fundamental operations supported by the PaaSManager.

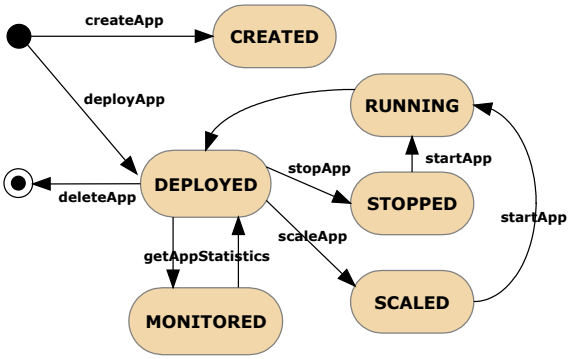

Fig. 2: Application Lifecycle

Besides these operations, the PaaSManager allows to restart, update, migrate applications and create and delete databases. As aforementioned, the portability of applications is also a main goal of this solution. The decision of migrating software to another provider may be based on the demand for better performance or more suitable business models. However, the PaaSManager approach does not intend to define any standard model or support interfaces to migrate, for instance, Java to .NET. Consequently, it requires a prior analysis to check if the new platform will be able to run the application properly. In terms of monitoring, each vendor provides distinct metrics and paradigms. CloudBees and Heroku have partnered with NewRelic for supplying real-time statistics. On the other hand, CloudFoundry and IronFoundry have a specific operation in their native APIs.

In the next section, the PaaSManager architecture will be presented as also all the modules that were designed for supporting the discussed features.

1) PaaSManager Architecture: The PaaSManager architecture is entirely modular so each vendor API has been implemented by different entities that abstract the background processes through a RESTFul interface. Figure 3 presents the framework architecture and respective operational modules.

2) PaaSManager API: The PaaSManager API is REST compliant supporting all the operations aforementioned in a lightweight and web oriented approach. This interface can be easily implemented by any HTTP client, such as, web, desktop or cURL. To be authenticated, the client application only needs to send an *api-key* through all the requests to the API. On the other side, the authentication with vendors is done through a unique account enabling the PaaSManager to act as a mediator between users and PaaS providers. Therefore the developer does not need to register in each vendor for having full access to their services.

3) Management Resources: The Management Resources presented in Figure 3 is a decision module responsible to interact with each PaaS adapter for management tasks (create, deploy, start, stop, etc.). Four adapters were defined, each one for each supported platform: CloudBees Mgmt, CloudFoundry Mgmt, IronFoundry Mgmt and Heroku Mgmt. These adapters implements the operations related to management features and exposed by the vendors APIs. Then, they process the acquired information returning unified responses in JSON or XML representations. Other key elements from the architecture are integrated with the Management Resources and the PaaS adapters. For instance, a central database keeps state of created applications storing the application framework identifier and the vendor name where the application is hosted. Also a Git server maintains all the applications source code repositories that are crucial for some PaaSManager operations, such as, create, deploy, update and migrate applications. Maintaining Git repositories may question the scalability of this architecture, however, to keep the system clean and efficient, the deletion of applications will enforce the removal of unnecessary repositories.

The migration of applications is becoming crucial in developers' point-of-view. In the scope of this work, the PaaSManager performs a prior analysis whether the platform to where the application will be migrated, supports the required technologies for the application to run properly. Figure 4 resumes the migration process and all the performed steps. Firstly, the request done to the PaaSManager API is routed to the Management Resources module, responsible for the migration operation. The database is then queried in order to return the PaaS identification where the application is hosted, as also the supported framework, e.g, Django, Sinatra. The Management Resources calls the specific PaaS adapter which in turn analyzes if the new provider supports the fundamental dependencies. In the case of correlation, the application code is deployed in the new selected PaaS through the required operations processes (create and deploy). Finally, the application deployed in the previous platform is removed and the state information in the database is updated.

Currently, the PaaSManager does not support the migration

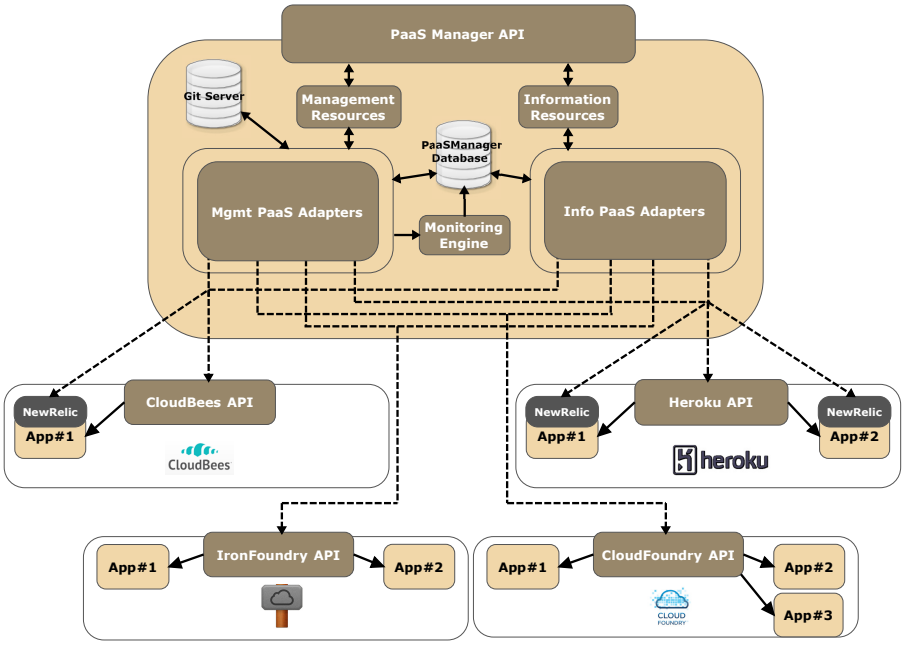

Fig. 3: PaaSManager Architecture

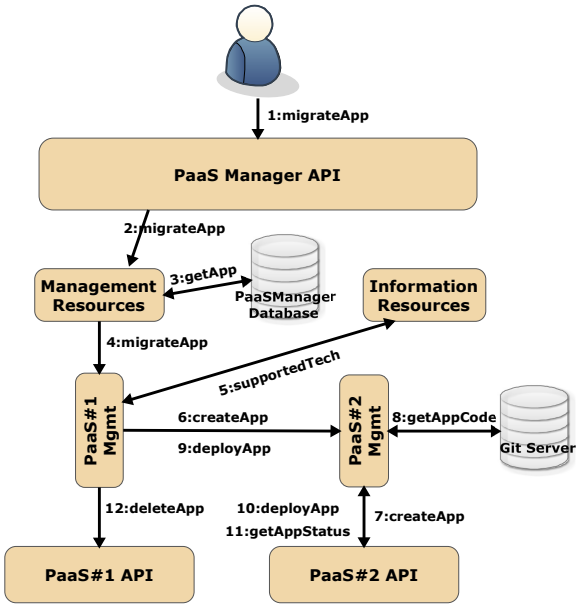

Fig. 4: Migration Process

of applications that require persistence, such as, databases. The limitations of such case would include the synchronizing of data between the two PaaS providers as well as the auto-reconfiguration of the application source code files in order to access the newly created database. However, the PaaSManager has a method that returns the database's access credentials for developers import or export data from the cloud.

4) Information Resources: The Information Resources presented in Figure 3 is a decision module that implements methods related to acquisition of information concerning applications and databases. It also offers four PaaS adapters: CloudBees Info, CloudFoundry Info, IronFoundry Info and Heroku Info, which are responsible for acquiring and processing information related to status, logs or monitoring tasks exposed by each vendor API. This information is fundamental for developers to manage their software and activate the scaling or migration processes.

5) Monitoring Engine: In recent years, some studies were conducted in cloud monitoring area aiming to define monitoring frameworks or metrics models for an efficient cloud management [10]. The metrics list is quite extensive, including availability, response time, RAM, CPU usage, database requests, threads or user sessions [11]. However at the moment, each PaaS provider offers different metrics and different tools for monitoring applications. As discussed previously, CloudBees and Heroku have partnered with NewRelic, which is a popular Application Management Performance used in cloud environments. On the other hand, CloudFoundry and IronFoundry supply a monitoring operation through their native APIs. The Monitoring Engine, presented in Figure 3, was developed in order to collect real-time metrics exhibited by each platform. After the application has been deployed in one of the PaaS, a background job is launched and kept alive until the application is stopped or until it is removed. This process is defined by a synchronous sampling performed every minute towards the NewRelic API or the native API according to the platform where the application is hosted. The achieved information is then stored in the central database and can be queried through the PaaSManager API.

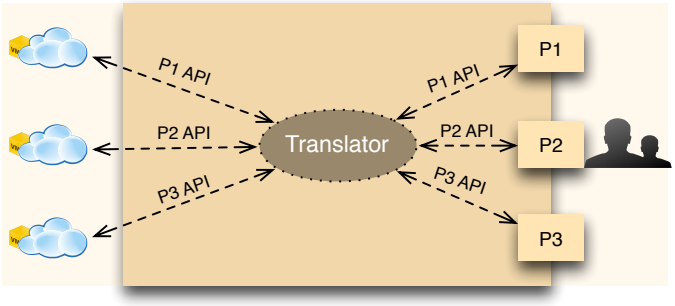

Fig. 5: IaaS Management: the 2-way translator

B. IaaS Manager

As mentioned in Section II, relying entirely on the success of the PaaSM doesn't cover all cloud customers' needs. A failed attempt to migrate an application to a cloud, fallbacks to the lowest cloud layer and results in the need for carrying out an endless low-level set of system administration tasks: provision the virtual machine, install the operating system, install and tweak user's application software dependencies, install security hardening configurations, add monitoring support, etc [12]. These are not the optimal sysadmin actions to be done. In our proposed solution an external entity could help distressing the workload by brokering the relationship between the two worlds. The IaaS Manager (IaaSM) acts as such an external entity.

The IaaSM is composed by three public APIs: IaaS Discovery Interface, IaaS Management Interface, and IaaS Functional Interface. The use of these APIs concedes users consolidated interfaces they can use, despite each cloud providers API variations.

1) IaaS Discovery Interface: The role of the IaaS Discovery Interface (IDI) is to bring forth all the available IaaS cloud providers, their functionalities, geographical location, and service fees. This implies a good entry-point to sort out which provider best suits the customer interests.

2) IaaS Management Interface: The IaaS Management Interface (IMI) unifies all IaaS providers APIs into one single API, abstracting end-users from the differences between clouds and so easily achieving management interoperability. A high-level management interface will provide a full range of operations such as create, start, stop, reboot, and terminate instances, upload and delete images, administer public and private IP addresses, etc. It is imperative to endorse users whom were forced by cloud providers to use proprietary and non-standard compliant interfaces when they moved in to the cloud. An 2-way API translator (Figure 5) equipping users to keep using unchanged tools against a second provider is advisable. Because CIMI helps attain interoperability between cloud service provider APIs and users, it is used as the foundation for the 2-way API translator.

3) IaaS Functional Interface: Once a cloud infrastructure has been created and deployed, users can benefit from the IaaS Functional Interface (IFI) to automate IT operations, from software provisioning and configuration to patching. Utilizing IFI to control how the system gets configured throughout its lifecycle, ensures an automated rebuild of the system in case of disasters or quickly deploys a mirror of the current system state.

C. Cloud Service Broker

The Cloud Service Broker (CSB) is the most important component on this architecture as it connects the PaaSM, IaaSM and user interfaces through a Service Bus. It provides extra high-level services that support all operation and business logic associated to the proposed solution. While the PaaS Manager and the IaaS Manager are self-contained, CSB orchestrates, further augments and eases the process of deploying and running an application in the cloud from the ground up, by advising users which platform better matches their application dependencies. At the same time it enables additional savings in operational expenses by deploying the code to the chosen one, and provisioning the proper monitoring hooks. Also through the use of the CSB, users are qualified to effortlessly migrate their applications from on-premises to the cloud or between cloud providers, with as few modifications as possible. Upon user request, the CSB will auto-scale possible computational resources on the platforms where the application is hosted, in order to accommodate high demand periods in a resilient manner. After peak hour or when demand simply drops, applications can be migrated back to its normal state, in a lower cost cloud provider.

It is known cloud offerings can significantly differ on supported functionality. For instance, there are PaaS services that do not yield monitoring or logging information. Others, while committed to assure no cloud confinement exists, it is only possible to interoperate with another provider if they share the same API which at present implies sharing the exact same platform. The CSB has a database containing data of provider capabilities, including functional, technical and location data. This knowledge is highly import for discerning which characteristics some providers support but others do not and provide means to better advise users with a filtered set of providers matching their application requirements. Section III-C1 further details and demonstrates how this data can be used to users' advantage.

The implementation of the CSB is not narrowed to the greatest common denominator of functionality from supported providers. In case of unsupported attributes from one or more providers, the CSB does not discard those features from all other supported providers. On the web portal, depending on where the application was deployed to, the user will be presented with a richer or fewer set of actions available to their disposal.

1) Manifest and Decision-making assessment: A key component of the system proposed is the concept of a manifest, and its relation to users and cloud providers. A manifest is a structured document delineating application minimal and optimal requirements, in the form of computation, storage, communication, and business metrics. The manifest is supplied

by the developer through the existing interfaces (web portal or CLI). As an example, the content of a manifest can specify that storage latency should not exceed 1ms, that bandwidth required is of 100Mbits, or simply that the application should not leave a given jurisdiction (or even the local premises). Therefore, it effectively consists in the description of the desired Service Level Agreement (SLA) requested to the PaaS provider. Users create a manifest document for each application, and submit it to the CSB. The document is evaluated, and is used during the entire application lifetime. At the first, it will enable the CSB to decide where to deploy the application. Later it enables the CSB to know which metrics must be monitored, the thresholds set by the application, and how to react if values exceed the given thresholds. The manifest response should therefore allow rating cloud offerings, returning individual scores for each application in each PaaS provider.

2) Deployment: Along with the source code, users are asked to inform where the application should be deployed to. The CSB collects the code, creates a revision control and source code management repository and copies the data into it. Based on the manifest, the CSB will request the creation of databases on the desired PaaS and will replace environment variables with the access credentials returned in a temporarily repository, leaving untouched the code previously uploaded to the repository. The rest of the deployment stages will occur transparently to the end-user, taking place in the PaaS Manager and/or the IaaS Manager. Auto redeploy of a new version to the cloud is just a matter of the user submitting changes to the revision control repository and the CSB will take care of everything else from there on.

3) Monitoring and Billing: The monitoring system is of the foremost importance to the CSB. The monitor audits computational resources usage per application and assembles daily, weekly and monthly reports allowing users to control costs of operation over time and covering all supported cloud platforms. Notifications will be sent out in case of events–thresholds exceeded, service outages, etc. Since the broker is a multi-tenancy framework, it will issue invoices per customer, not per application. Monitoring is also important for the CSB to analyze if the requirements specified in the manifest are being fulfilled, and to rank existing providers according to the needs of each application.

4) Migration: By leaving the source code untouched and with the manifest stored on the CSB, future deployments to concurrent providers will turn out to be a simplified process to both user and broker. Users won't need to change credentials nor host names. They will continue to access the service through the same domain assigned when first deployed to the CSB. The CSB will ensure the synchronization of persistent data without succor to a noticeable downtime period.

IV. Conclusion and Future Work

With the growing number of cloud providers, the search for the best platform to deploy and manage applications is a critical factor and may not always lead to the best decision. This paper addresses the aforementioned inconvenients proposing a high-level comprehensive architecture intended to ease application lifecycle stages, such as, deployment, management, and monitoring of cloud applications over the two cloud models PaaS and IaaS.

The proposed CSB is a key part from the framework by delivering and evaluating, the most appropriate platform from a catalog of miscellaneous PaaS offerings, based on the application profile or defined cost thresholds. However, if the supported platforms don't cover all the users' needs, the developer shall be forwarded and assist in preparing and setting up a PaaS on demand settled on lowest cloud layer solutions. This approach opens a bond between PaaS and IaaS which is orchestrated via the CSB and implemented through the outlined PaaS Manager and IaaS Manager.

An example of market players who can take advantage of such multipurpose cloud framework are the communications service providers (CSP). With the new added-value service providers, operators are becoming just a data-pipe guarantying connectivity between both ends. CSPs are undoubtedly interested in taking a share of the growing cloud market, setting a major position. The exploitation of two-sided business revenues with third-parties developers toward the management and migration of applications to private or hybrid cloud products is a likely model.

As proof-of-concept, a first prototype of the CSB and PaaS Manager is under development as well as the user's web portal and command line with Git integration. The monitoring system for covering resources usages and costs notifications will be tested soon for running intensive tests and benchmarks through real applications.

Acknowledgment

The work hereby presented is a Portugal Telecom Inovação funded project conducted under the ATNoG research group, Instituto de Telecomunicações (Aveiro pole), Portugal.

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, 2010, 10.1007/s13174-010-0007-6. [Online]. Available: http://dx.doi.org/10.1007/s13174-010-0007-6

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep., 2009.

[3] "The nist definition of cloud computing," http://www.csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011, [Online; accessed 9 May 2012].

[4] D. Collison, "Distributed Design and Architecture of Cloud Foundry," http://www.slideshare.net/derekcollison/design-of-cloud-foundry, 2011, [Online; accessed 9 May 2012].

[5] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier, "A Federated Multi-Cloud PaaS Infrastructure," in *5th IEEE International Conference on Cloud Computing*. hawaii, États-Unis: IEEE Xplore Digital Library, Jun. 2012. [Online]. Available: http://hal.inria.fr/hal-00694700

[6] Z. ur Rehman, F. Hussain, and O. Hussain, "Towards multi-criteria cloud service selection," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, 30 2011-july 2 2011, pp. 44 –48.

[7] J. K. J. Kang and K. M. S. K. M. Sim, "Cloudle: A multi-criteria cloud service search engine," pp. 339–346, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5708589

[8] T. Liu, Y. Katsuno, K. Sun, Y. Li, T. Kushida, Y. Chen, and M. Itakura, "Multi cloud management for unified cloud services across cloud sites," in *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, sept. 2011, pp. 164 –169.

[9] M. Maiya, S. Dasari, R. Yadav, S. Shivaprasad, and D. Milojicic, "Quantifying manageability of cloud platforms," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, june 2012, pp. 993 –995.

[10] J. Shao and Q. Wang, "A performance guarantee approach for cloud applications based on monitoring," in *Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, ser. COMPSACW '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 25–30. [Online]. Available: http://dx.doi.org/10.1109/COMPSACW.2011.15

[11] X. Cheng, S. Yuliang, and L. Qingzhong, "A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA," in *Joint Conferences on Pervasive Computing*, ser. JCPC 2009. Tamsui, Taipei: IEEE Internet Computing, 2009, pp. 599–604.

[12] T. Wood, A. Gerber, K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "The case for enterprise-ready virtual private clouds," in *Proceedings of the 2009 conference on Hot topics in cloud computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855537