A Comparison Framework and Review of Service Brokerage Solutions for Cloud Architectures

Frank Fowley[1], Claus Pahl[1], and Li Zhang[2]

[1] IC4, Dublin City University, Dublin 9, Ireland
[2] Northeastern University, Shenyang, China

Abstract. Cloud service brokerage has been identified as a key concern for future cloud technology development and research. We compare service brokerage solutions. A range of specific concerns like architecture, programming and quality will be looked at. We apply a 2-pronged classification and comparison framework. We will identify challenges and wider research objectives based on an identification of cloud broker architecture concerns and technical requirements for service brokerage solutions. We will discuss complex cloud architecture concerns such as commoditisation and federation of integrated, vertical cloud stacks.

Keywords: Cloud Broker; Service Brokerage; Architecture Patterns; Cloud Broker Comparison; State-of-the-art Review; Research Challenges

1 Introduction

Several organisations active in the cloud technology area, such as Gartner and NIST [12, 20], have identified cloud service brokerage as an important architectural challenge. Architecture and programming model concerns are key enabler of any service brokerage solution that mediates between different providers by integrating, aggregating and customising services from different providers. We compare cloud service management and brokerage solutions, i.e. we discuss a broader classification in terms of components and features of cloud service brokers, specifically looking at architecture, language and quality as technical aspects in a refined, more descriptive model. We address challenges based on an identification of cloud broker architecture patterns for service brokerage solutions. Our key contribution is a discussion of service broker solutions based on a 2-pronged comparison framework. Such a dedicated framework does not exist for cloud brokers and goes beyond existing service taxonomies such as [13].

The paper is organised as follows. Cloud service brokerage is introduced in Section 2. Section 3 discusses wider architectural concerns. In Section 4, we introduce and apply the comparison framework. These investigations lead into a broader research challenges discussion in Section 5.

2 Cloud Service Brokerage

Gartner and NIST define Cloud Service Brokerage [12, 20]. They follow a similar three-pronged classification. They define a cloud broker as an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers [11].

In this overview of key concepts, we follow Gartner. Aggregation is actually singled out by both organisations. NIST intermediation and Gartner customisation focus on enhancing existing service. NIST arbitration and Gartner integration have in common a flexible mediation and integration of different systems.

- Aggregation is about delivering two or more services to possibly many consumers, not necessarily providing new functionality, integration or customisation, but offering centralised management of SLAs and security.
- Customisation is about altering or adding capabilities, here to change or improve and enhance the service function, possibly combined with analytics.
- Integration addresses the challenges of making independent services work together as a combined offering, which is often integration of a vertical cloud stack or data/process integration within a layer. Classical techniques such as transformation, mediation and orchestration are the solutions.

We now look at the possible impact of the different cloud layers IaaS and PaaS on cloud service broker requirements. Brokers have to deal with various cloud layer-specific concerns [4], e.g. for IaaS these are:

- The key IaaS need is elasticity. With techniques such as replication, provisioned services can be scaled. Images can be replicated and moved to other, interoperable offerings and platforms to create a virtual layered environment.
- Problems that arise are that platform engines are often proprietary or do not replicate fully unless standards like OVF for VMs are used. Also, replicating an image with data needs bandwidth, which requires optimised solutions.
- Image and data handling aims to minimise replication and manage deletion, use segmentation for services, differentiate user-data and images/services to optimise and include intelligent data management such as map-reduce techniques. Horizontal scaling often requires the full dataset to be replicated. Vertical scaling can be based on data segmentation and distribution.

This indicates that automation is here of critical importance as the cloud elasticity need is the driver of these techniques. For the PaaS layer:

- Platforms need to facilitate composition and service mashups [10, 3].
- For most applications, base image duplication can suffice, but too many users per application generally require full replication with customer-specific data/code. In case base images (e.g. for .NET) are available, we only need to replicate service instances, but not a full image.
- Further problems arise for composition as QoS is generally not compositional, e.g. the security of a composition is determined by its weakest link.

Automated management is a key concern. Standardisation in terms of OVF as an image format or OCCI as an interface for infrastructure-level resource management functionality are solutions. Some solutions exist IaaS standards like OCCI and CIMI cover service lifecycle management, TOSCA addresses portability and

CDMI is about data management. IaaS open-source systems supporting these standards are Openstack, which is a lifecycle management product in the line of CIMI and OCCI aims, or the mOSAIC API that supports composition and mashups at an infrastructure level [19]. PaaS systems include Cloudify, a management tool for vertical cloud stack integration, and Compatible One, a broker for horizontal integration [7, 8].

3 Cloud Service Broker Architectures

Cloud brokerage solutions build up on existing virtualisation, cloud platform and IaaS/PaaS/SaaS offerings. We can single out three architecture patterns:

- Cloud Management: supports the design, deployment, provisioning and monitoring of cloud resources, e.g. through management portals. This is an extension of the core lifecycle management (LCM), adding monitoring features or graphical forms of interaction. Rudimentary features for the integration of compatible services can be provided.
  A management layer is often identified in cloud architecture to management that facilitate efficient and scalable provisioning in a number of the platforms reviewed below.
- Cloud Broker Platform: supports the broker activity types discussed earlier – aggregation, customisation, integration – which needs a specific language to describe services in a uniform way and to define the integration mechanism. The origin of this is the common broker pattern from software design patterns, applied to a cloud setting.
- Cloud Marketplace: builds up on broker platform to provide a marketplace to bring providers and customers together. Again, service description for core and integrated services plays a role for functionality and technical quality aspects. Trust is the second key element that needs to be facilitated.
  Marketplaces for apps are omnipresent and this marketplace pattern is a reflection of upcoming cloud-specific marketplaces (DT will be mentioned as a sample case below).

These layers can be put on top of the classical cloud architecture layers SaaS, PaaS and IaaS. The discussion below will show that a fine-grained characterisation of cloud brokerage solutions, even beyond these three is necessary to identify and distinguish specific challenges. We look into open-source solutions (or solutions provided by publicly funded projects) as these are well-documented. Open-source solutions can thus be categorised based on the presented scheme:

- Open IaaS: OpenStack, for instance, is a basic IaaS cloud manager that transforms data-centres to become IaaS clouds [24].
- Open PaaS: OpenShift and CloudFoundry are open PaaS platforms assisting the cloud app developer by commoditising the software stack [6, 23].

The Open IaaS/PaaS solutions can be differentiated from respective IaaS/PaaS brokers. In the following, we will try to point out the salient differences between

	CATEGORIES and TYPE					
Name	Category	Cloud Layer	Multi Cloud API Library	IaaS Fabric Controller	Open PaaS Solution	Open PaaS Provider
OpenNebula	CLOUD FABRIC CONTROLLER	IaaS		Y		
OpenStack	CLOUD FABRIC CONTROLLER	IaaS		Y		
libcloud	API LIBRARY	PaaS	Y			
jclouds	API LIBRARY	PaaS	Y			
simpleAPI	API LIBRARY	PaaS	Y			
DeltaCloud	API SERVER	PaaS	Y			
Cloudify	CLOUD DEVOPS & LCM	PaaS	Y		Y	
Mosaic	PAAS	PaaS	Y		Y	
Cloud Foundry	PAAS	PaaS	Y		Y	Y
OpenShift	PAAS	PaaS			Y	
CompatibleOne	IAAS BROKER	PaaS			Y	
4Caast	SERVICE BROKER	PaaS				
Optimis	IAAS BROKER	PaaS			Y	

Table 1. Open Source Clouds - System Category and Type.

some cloud brokers that go beyond IaaS/PaaS management solutions. Optimis and CompatibleOne are IaaS-oriented, and only 4CaaSt targets PaaS and to some extent also the SaaS domain. There is, however, SaaS broker activity in the commercial space.

An observation here is that the broker pattern receives attention and that reusable solutions are in development, starting with the IaaS layer, but including IaaS and PaaS over time. The existence of marketplaces, which are interesting for the diverse SaaS space, indicates the existence of broker solution. The AppDirect commercial broker is an example. However, a wider range of commoditised, ready-to-use broker platforms can be expected in the future – to service the different broker types defined, but also provide a fuller range of features as our discussion of the open-source solutions indicates.

4 Service Management and Brokerage Comparison

In this section, we compare cloud solutions using a dedicated 2-pronged framework, which we will introduce first.

- The first is a categorisation schema for a basic classification (Tables 1, 2).
- The second is a more detailed, descriptive classification (Tables 3 to 5).

We compare a number of selected solutions, essentially open-source solutions or publicly funded frameworks.

In Tables 1 and 2, we categorise a number of solutions [6, 7, 9, 15, 17, 22, 19, 23, 24, 33, 1, 8]. We categorise languages in terms of the cloud layer support, but also specific features or functions each of them provides. We have defined a comparison framework to categorise solutions along the following concerns:

- System Type: Multi Cloud API Library, IaaS Fabric Controller, Open PaaS Solution, Open PaaS Provider.
- Distribution Model: Open Source (for all solutions considered).

	CORE CAPABILITES			CORE FEATURES						ADVANCED FEATURES		
Name	Multi IaaS Support	Multi Language / Multi Framework	Multi Stack	Service Description Language	Native Data store	Native Message Queue	Programming Model	Elasticity Scalability	QoS / SLA Monitoring	Service Discovery / Composition	Broker	Market-Place
OpenNebula												
OpenStack					Y							
libcloud	Y											
jclouds	Y											
simpleAPI	Y											
DeltaCloud	Y											
Cloudify	Y	Y	Y	Y			Y	Y	Y			
Mosaic	Y			Y	Y	Y	Y	Y		Y	Y	
Cloud Foundry	Y	Y	Y				Y					
OpenShift		Y	Y					Y				
CompatibleOne	Y			Y			Y	Y	Y	Y	Y	
4Caast				Y	Y	Y			Y	Y	Y	Y
Optimis				Y	Y		Y	Y	Y		Y	

Table 2. Open Source Clouds - Core Capabilities and Features/Components.

- Core Capabilities: Multi-IaaS Support, Multi Language / Multi Framework Support, Multi Stack Support.
- Core Features/Components (development and deployment time): Service Description Language, Native Data Store, Native Message Queue, Programming Model, Elasticity & Scalability, QoS/SLA Monitoring.
- Advanced Features/Components: Service Discovery/Composition, Broker, Marketplace – towards broker and marketplace features.

We chose these concerns to, firstly, broadly categorise the solution in terms of is main function (the system type that indicates its target layer and central function in that layer) and whether it is proprietary or open-source. Secondly, a range of standard properties and individual components are singled out. Properties chosen here (the Core Capabilities) refer to necessary capabilities for brokers to integrate offerings. The two features categories organised a number of system components into common and more advanced ones.

In the following we review various solutions with respect to three facets: architecture & interoperability, languages & programming, and quality. This format allows us to drill down and compare using a more descriptive format. We will not consider all 13 products initially compare, but only select the most advanced ones for each aspect. This second, deeper and more descriptive classification schema is based on three facets.

Architecture and Interoperability. The solution architecture is a key element in the definition of a broker. Of practical relevance are the existing, typically lower-layer solutions that the system supports. This is an interoperability concern. CompatibleOne is OCCI-compatible in its support for VM management. For instance, Mosaic assumes a Linux OS, which runs Mosaic App Components (called CloudLets). A number of common commercial cloud solutions are supported by Mosaic, including Amazon and Rackspace products.

In Table 3, a number of PaaS-level solutions are summarised in terms of these two aspects. Common are the utilisation of configuration management solutions, such as Chef or GIT. The deployment is managed through consoles

	Cloudify	*CloudFoundry*	*OpenShift*	*Compatible1*	*4Caast*
Architecture	- Console for platform commands. Web management console for monitoring. - Service Manager uses scripting (recipe) to cater for middleware stack - Cloud Controller is REST endpoint to manage app deployment & control; injects agent on VM to install & orchestrate app deploy/monitor/ scale - Cloud Driver: VM templates for different IaaS clouds in configuration. Triggers host provisioning	- Console pushes app to cloud; deployment management / configuration through console - Controller runs as a cloud VM on the target IaaS; controls all Cloudify spawned cloud VMs. Does not manage IaaS layer functions. The IaaS provider must support Cloudify. Apps created using Cloudify are deployed to Cloudify VMs controlled by a cloud controller on Cloudify-compliant IaaS clouds.	- Divided into control plane (Broker) and messaging / application hosting infrastructure (Nodes). - Controller is command CLI shell, used to create apps. GIT for app management / deployment. - Gear is application container and a virtual server/node accessed via ssh. Cartridge service runs on a Gear. App LCM scripts allow for post-deployment action hooks to run on VMs.	- ACCORDS exposes features through REST API. - Parser validates Manifest against CORDS schema and maps elements to valid OCCI categories which are then instantiated. - Publisher provides which endpoint serves which categories. Parser runs and produces a plan of OCCI instances for resolution (instance can receive/send data). - Broker processes plan and invokes instances.	- Execution Container REC runs instances. - Deployment Manager maps deployment model (service template, QoS constraints) to OVF. Service Manager deploys images using Claudia. - REC includes an agent (application LCM, control) and a server (storage, config data). Deployment Server (Chef) talks to Service and REC Manager. OVF Manager creates extended OVFs from abstract resolved BluePrints.
Clouds Supported / Interoperability	Supports Azure, OpenStack, CloudStack, EC2, Rackspace, Terramark (buildable for any of the jclouds above)	Supports AWS, vSphere, Openstack, Rackspace. Is hosted as public PaaS on Cloudify. Private cloud is available.	Uses DeltaCloud; app runs on RedHat certified public cloud (needs deltacloud support).	OCCI provider interfaces (PROCCIs) for OpenStack, OpenNebula and Azure (also SlapOS and SlapGrid).	FlexiScale driver provided. OpenNebula supported. Generic IaaS Cloud API through Tcloud.

Table 3. Architecture and Interoperability.

or APIs, mapping PaaS-level requests down to IaaS operations. As often many IaaS solutions are supported, interoperability is a critical concern.

Languages and Programming. Service description plays a key role for interoperability [27]. For selected solutions, we look at the following three aspects:

– service language – the core notation, including the coverage of concerns vertically (PaaS/IaaS integration) and horizontally (full lifecycle management) and how this is manipulated (format and API).

– programming model – using the language to program brokerage solutions, linking to SOA principles and other development paradigms.
– service engineering – covering wider design and architecture concerns, including monitoring and mashups.

Cloudify, for instance, uses application recipes and resource node templates in the form of Groovy scripts as the programming model. A service recipe contains LCM scripts, monitoring probes and IaaS resources requirements. Mosaic uses an OWL ontology as the notation and a component-based application programming model for portability of apps across Mosaic-compliant clouds. More solutions are compared in Table 4. Patterns emerge as solutions to compose, connect and manage clouds in distributed contexts.

Quality. Scalability and elasticity are specific cloud concerns, and need to be addressed by the service description notation. Load balancers are typically used to control elasticity based on monitored key performance indicators (KPIs). Multi-tenancy, if available, can alleviate elasticity problems. Based on specifications, these are looked after by configuration management tools to set up probes and monitoring tools to collect and analyse data. Table 5 covers these concerns.

Summary. We can categorise the open-source solutions based on some of the central aspects. This summarises a selection of currently available solutions in terms of their support aims and allows us to identify trends. Developers are supported in three categories: a) API library: libcloud, Jcloud, deltacloud, b) Devops: Cloudify and c) Full PaaS: CloudFoundry, OpenShift. A trend goes from provider-oriented solutions to developer-oriented solutions to end user-oriented cloud management [2] – 4Caast being an example of the latter.

5 Challenges – Brokers, Markets and Federated Clouds

The need for interoperability becomes apparent in the context of cloud service brokerage, where independent actors in the ecosystem integrate, aggregate/compose and customise/adapt existing services [12, 20]. End-to-end personalisation becomes achievable. Prosumers create mashups from existing services.

From the above comparison between various cloud solutions, we can note a difference between the needs of cloud brokerage and cloud marketplaces. We did already introduce them as different patterns above.

– The *brokerage* needs to automate as far as possible the process of matching service requirements with resource capacity and capabilities [32]. The ideal would be a total commoditisation of IaaS so that any compute resource, be it from a private OpenStack cloud or a public EC2 instance, could be plugged into a user's compute capacity. Therefore, interoperability will remain of importance. In this regard, it is useful to look at new areas of compatibility that should be considered in matching that are not handled by brokers currently. For example, none of the three open-source solutions that were assessed considered data integrity as a matching criterion; however, they all included performance in their criteria. Security policy is another aspect that has a

	Reservoir	*Compatible One*	*4Caast*	*Optimis*
Service Language	Service Definition Manifest for metadata; software stack (OS, middleware, app, config, data) in a virtual image; has service descriptions for contracts between service provider SP and infrastructure provider IP. Manifests (OVF) relate abstract entities and LCM / operation of services. Feedback between SP and IP allows IP to scale and monitor.	Units of Service Manifest: Image & Infrastructure. Image: System (base OS) & Package (stack config); Infrastructure: Storage, Compute & Network. Image is description of manual app build. Image has agent that is embedded in VM & runs on startup. Agent is script to run required configuration, set up monitoring probes, or download components.	Resources and Services are described in a Blueprint BP, which is an abstract description of what needs to be resolved into infrastructure entities. BPs are stored and managed in a BP repository via a REST API. A BP is resolved when all requirements are fulfilled by another BP, via the Resolution Engine (is service orchestration feature).	Service Manifest includes sections per component per VM. Service Register has sections for SP requirements and IP capabilities, VM abstract description, TREC (trust, risk, eco-efficiency, cost), elasticity, data protection. Optimis also provides a cloud provider description schema for a SP to provide its capabilities in an XML Optimis-compliant format.
Programming Model	Elasticity is defined using ECA rules to scale infrastructure dynamically based on application KPI metrics. Rules in OCL.	PaaS4Dev: Java EE services (EE5/6 web profile) & Enterprise OSGi services (http, jndi, transaction) for development	Uses Active MQ, postgresql, jonas, ow2orchestra, apache serv bus. Ontology-based BP schema using Jena, SPARQL.	Java schemas, jaxb, xmlbeans, REST, monitor; also jax-ws, cxf, javagat. IDE is Eclipse with plugin for Optimis core classes.
Service Engineering	Service provisioning described in Deployment Descriptor. Service configuration automation based on Xen configuration. Service Elasticity is achieved through mapping Manifest KPIs with run-time metrics gathered by app monitoring agents.	- Nested manifests support service composition. - COSACS module embeds in VM image mechanisms to manage lifecycle actions, e.g. post-creation monitoring setup and appliance configuration, in conjunction with image production module.	- Request Language BRL & request patterns create Blueprint BP service specification - mapped to cloud operations and cloud mgmt API calls. Mashup for composition. - BP consists of BP images, contains functional, KPI & policy parameters.	Toolkit provides image mgmt, context manager injects context information to VMs and Elasticity Engine to add/remove resources. Service Deployment Optimiser optimises placement of services. Configuration using the Toolkit IDE.

Table 4. Service Language, Programming Model and Service Engineering.

technical nature, but is also abstract insofar as it can be implemented by a cloud provider. Data integrity and security policy enforcement, if considered as criteria when evaluating cloud interoperability, may need to be formalised using a language to describe common aspects, similar to the languages that have been created to model other cloud entities.

	CloudFoundry	*OpenShift*	*Compatible1*	*4Caast*	*Optimis*
Elasticity / Scalability	Can add/remove instances for scalability and increase/ decrease CPU & memory limits on VMs	- Gears automatically added/removed as load changes - Multi-tenancy efficiency using multi-gears on same VM	Elasticity is provided by the load balancer module for the IaaS resources.	Not in current release.	The toolkit includes an Elasticity Engine to add / remove resources.
QoS / SLA Monitoring	There is only a basic logging facility with Cloud foundry but there are many third-party Cloud Foundry monitoring plugins can be used to provide application monitoring, such as Hyperic.	The application scaling, when automatic, is based on concurrent application request thresholds. The amount of resources consumed by an application can be monitored and viewed from the Console.	via COMONS Monitoring module.	Monitoring based on probe injection on PICs via REST. Modified JASMINe framework provides dynamic probe deployment & config. Chef recipe configs VM probes to be used by REC manager. Monitoring is based on collectd stats for forecasting.	Framework uses REST to get CPU/disk usage from monitoring. Monitors reside on logical/physical nodes and run as scripts to feed data to monitor store. SLA Manager built using WSAG4J is implementation of OGF WS-Agreement standard.

Table 5. Quality: Scalability/Elasticity and SLAs.

– The *marketplace* will need to additionally focus on the architecture of the applications as well as the cloud. The appstore model appears to be the defacto model of choice for the marketplace, but this seems more an admission of the success of the Apple initiative rather than any research. There may be a potential to explore other forms of the online marketplace suitable to cloud apps and their composition [18, 10]. This could also be pushed to an even more commodity-based scenario where all services could be registered on a wide-area multi-marketplace scale facilitating an even greater eco-system.

Commoditisation. The commoditisation of cloud services is an emerging need from the discussion above – specifically from the language and programming facet. A trend is to move from the lower IaaS layer to PaaS and onwards to encompass SaaS, aiming to integrate lower layers – 4CaaSt is an example. To make this work, services at all layers need to be available for a uniform way of processing in terms of selection, adaptation, integration and aggregation. Commoditisation is the concept to capture this need. Some concrete observations related to the reviewed three open-source solutions are: fully functional image and vertical stack building capabilities (CompatibleOne leadership), operational support of service composition (4CaaSt leadership) and graphical manipulation of service abstractions (Optimis leadership). Facilitated can commoditisation be through a uniform representation through description templates such as recipes, manifests or blueprints. These need to cover the architecture stack and meet the language and quality concerns discussed in Section 4. Commercial providers are equally working on the commoditisation of cloud services as described above.

Commoditisation is an enabler of marketplace functions that sit on top of a broker. Thus, additional challenges and requirements for marketplaces are:

- Data integration and security enforcement as non-functional requirements.
- Social network functions allow service ratings by the communities.
- SLA management to be integrated, e.g. in terms of monitoring results.

Commoditisation needs to be facilitated through an operational development and deployment model. It therefore acts as an enabler. Trust is an equally important concern that more difficult to facilitate technically than commoditisation. A mechanism is needed for not only vetting individual providers, but also to allow this to happen in layered, federated and brokered cloud solutions.

In another direction, there has not been a proliferation of cloud capacity clearing-houses that would operate similar to a spot market to allow clouds to buy and sell spare cloud capacity on a very short-term basis. It is not clear what new areas of research would be needed to facilitate such a movement in the cloud. It seems reasonable that, with the continued commoditisation of the cloud by brokers and marketplaces, such a trend could be seen eventually.

Federated Clouds. Federation is the second requirement for brokerage solutions [5], i.e. to work across independently managed and provided cloud offerings of often heterogeneous nature. Challenges and requirements in this context arising from the architecture and interoperability discussion (the first facet) are:

- Reference architectures – e.g. NIST cloud brokerage reference architecture.
- Scope of control – the management of configuration and deployment based on integrated and/or standards-based techniques [16].
- Federation and syndication – as forms of distributed cloud architectures [31].

6 Conclusions

We have introduced the main concepts of service brokerage for clouds, using some concrete systems and platforms to identify current trends and challenges and compare current, primarily open-source solutions. Brokerage relies on interoperability, quality-of-service and other architectural principles. Brokers and marketplaces will play a central role for new adopters migrating into the cloud or between cloud providers [14, 28]. Brokers will act as first points of call.

A 2-pronged comparison framework is the first contribution where we provided a first categorisation scheme to characterise the solution in terms of type, common components and features. The second scheme is a more descriptive, layered taxonomy starting with architecture and interoperability, languages and programming, and quality as facets.

An observation of our comparison based on the framework is the emergence of cloud broker solutions on top of cloud management. A further separation of marketplaces, often in the form of appstores, is necessary. A number of activities work in this direction. Compatible One is a good example showing how OCCI is used as an infrastructure foundation and built upon to provider PaaS-level brokerage. 4CaaSt in a similar vein aims to integrate the layers and move toward a marketplace solution. Commercial solutions, such as DT and UShareSoft, show already existing brokerage and marketplace solutions ranging from images to software services, essentially commoditising the respective cloud resources.

Service description mechanisms discussed in [21, 30, 26] (in the form of manifests, recipes and blueprints) , but also in standards like TOSCA and CloudML, can serve to abstract, manipulate and compose cloud service offerings in an effort to commoditise the cloud. These description mechanisms, based on an abstract model serve two purposes: Firstly, to abstractly capture, present and manipulate cloud resources. Secondly, to serve as a starting point to link to configuration and other deployment concerns in federated clouds. Thus, commoditisation and federation emerge as challenges from our discussion.

Acknowledgments. This research has been supported by the Irish Centre for Cloud Computing and Commerce, an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

References

1. 4Caast. 4CaaSt PaaS Cloud Platform. http://4caast.morfeo-project.org/. 2013.
2. T. Benson, A. Akella, S. Sahu, A. Shaikh. Peeking into the Cloud: Toward User-Driven Cloud Management. CloudS 2010 Conference, Sydney, Australia. 2010.
3. D. Benslimane, S. Dustdar, A. Sheth. Services Mashups: The New Generation of Web Applications. Internet Computing, vol.12, no.5, pp.13-15, 2008.
4. D. Bernstein, E. Ludvigson, K.Sankar, S. Diamond, M. Morrow Blueprint for the Inter-cloud: Protocols and Formats for Cloud Computing Interoperability. Intl Conf Internet and Web Appl and Services. 2009.
5. R. Buyya, R. Ranjan, R.N. Calheiros. Intercloud: Utility-Oriented Federation of Cloud Computing Environments For Scaling of Application Services. Intl Conf on Algorithms and Architectures for Parallel Processing , LNCS 6081. 2010.
6. Cloud Foundry. Open Source PaaS Cloud Provider Interface. http://www.cloudfoundry.org/. 2013.
7. Cloudify. Cloudify Open PaaS Stack. http://www.cloudifysource.org/. 2013.
8. CompatibleOne. Open Source Cloud Broker. http://www.compatibleone.org/. 2013.
9. DeltaCloud. Deltacloud REST cloud abstraction API. http://deltacloud.apache.org/. 2013.
10. C. Fehling, R. Mietzner. Composite as a Service: Cloud Application Structures, Provisioning, and Management. Information Technology 53:4, pp. 188-194. 2011.
11. Forrester Research. Cloud Brokers Will Reshape The Cloud. 2012. http://www.cordys.com/ufc/file2/cordyscms_sites/download/09b57cd3eb6474f1fda1cfd62ddf094d/pu/

12. Gartner - Cloud Services Brokerage. Gartner Research, 2013. http://www.gartner.com/it-glossary/cloud-services-brokerage-csb
13. C.N. Höfer, G. Karagiannis. Cloud computing services: taxonomy and comparison. Journal of Internet Services and Applications, 2(2), 81-94. 2011.
14. P. Jamshidi, A. Ahmad, C. Pahl. Cloud Migration Research: A Systematic Review. IEEE Transactions on Cloud Computing. 2013.
15. Jclouds. jclouds Java and Clojure Cloud API. http://www.jclouds.org/. 2013.
16. A.V. Konstantinou, T. Eilam, M. Kalantar, A.A. Totok, W. Arnold, E. Sniblel. An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds. Intl Workshop on Virtualization Technologies in Distr Computing. 2009.
17. Libcloud. Apache Libcloud Python library. http://libcloud.apache.org/. 2013.
18. R. Mietzner, F. Leymann, M. Papazoglou. Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns. Intl Conf on Internet and Web Applications and Services. 2008.
19. Mosaic. mOSAIC Multiple Cloud API. http://www.mosaic-cloud.eu/. 2013.
20. NIST. Cloud Computing Reference Architecture. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505. 2011.
21. D.K. Nguyen, F. Lelli, Y. Taher, M. Parkin, M.P. Papazoglou, W.-J. van den Heuvel. Blueprint Template Support for Cloud-Based Service Engineering. Proceedings ServiceWave11, Poznan, Poland, October 2011.
22. OpenNebula. OpenNebula - Open Source Data Center Virtualization. http://opennebula.org/. 2013.
23. OpenShift. Cloud computing platform. https://openshift.redhat.com/. 2013.
24. OpenStack. OpenStack Open Source Cloud Computing Software. http://www.openstack.org/. 2013.
25. Optimis. Optimis - Optimized Infrastructure Services. http://www.optimis-project.eu/. 2013.
26. C. Pahl. Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture. European Conf on Model-Driven Architecture ECMDA2005. 2005.
27. C. Pahl, S. Giesecke and W. Hasselbring. Ontology-based Modelling of Architectural Styles. Information and Software Technology (IST). 1(12): 1739-1749. 2009.
28. C. Pahl, H. Xiong. Migration to PaaS Clouds - Migration Process and Architectural Concerns. IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA 2013. 2013.
29. C. Pahl, H. Xiong, R. Walshe. A Comparison of On-premise to Cloud Migration Approaches. Europ Conf on Service-Oriented and Cloud Computing ESOCC. 2013.
30. M.P. Papazoglou, W.J. van den Heuvel. Blueprinting the Cloud. IEEE Internet Computing, November 2011.
31. A. Paya, D.C. Marinescu. Clustering Algorithms for Scale-free Networks and Applications to Cloud Resource Management. 2013.
32. L. Rodero-Merino, L.M. Vaquero, V. Gil, F. Galn, J. Fontn, R.S. Montero, I.M. Llorente. From Infrastructure Delivery to Service Management in Clouds. Future Generation Computer Systems, vol. 26, pp. 2261240. 2010.
33. simpleAPI. Simple API for XML. http://en.wikipedia.org/wiki/Simple_API_for_XML. 2013.
34. L. Sun, H. Dong, and J. Ashraf. Survey of Service Description Languages and Their Issues in Cloud Computing . Eighth International Conference on Semantics, Knowledge and Grids (SKG) 2012. pp. 128-135. IEEE. 2012.