PaaSPort – A UNIFIED PaaS-CLOUD MANAGEMENT APPLICATION AVOIDING VENDOR LOCK-IN

Bernd Zwattendorfer, Bojan Suzic, Gabriel Schanner
Graz University of Technology, Institute for Applied Information Processing and Communications (IAIK)
Inffeldgasse 16a, 8010 Graz, Austria

ABSTRACT

PaaS (Platform as a Service), a service model in the cloud computing service market, specializes in offering a preconfigured infrastructure for developers, enabling them to deploy, host and scale applications efficiently. As the PaaS market continues to grow, new providers and more sophisticated services emerge. The great variety of service-specific offerings complicates the customers' abilities to choose a fitting solution efficiently. It furthermore prevents the switching of providers when necessary, making customers dependent on particular vendors. This issue is known as a *vendor lock-in*, often caused by impractical and economically unviable provider switch. To address the particular issue on vendor lock-in, in this paper we introduce PaaSPort – a unified PaaS-Cloud management application that enables the transparent and unified management of cloud applications at different PaaS providers. PaaSPort acts as an interface for various PaaS providers, enabling cloud customers to monitor and modify their PaaS hosted applications independently. All functionality of this interface application is exposed through a REST web API that unifies the common operations of multiple providers and allows for provider specific extension.

KEYWORDS

Platform as a Service, PaaS, cloud computing, management application, vendor lock-in, API

1. INTRODUCTION

In the last years different branches of the booming cloud computing service market have emerged, one of them being Platform as a Service (PaaS). PaaS primarily refers to a preconfigured and flexible computing platform environment. The goals of this technology are manifold and most focus on boosting the efficiency in the process of development, deployment and maintenance of applications in the cloud. This is accomplished by increasing programmer productivity, reducing costs, and generally allows for faster build and release cycles by stripping away low level infrastructure management and providing services as easily configurable and scalable components (Lawton, 2008) (Chieu et al, 2009).

As the cloud market continues to grow, with a prognosticated total worldwide revenue of $2.9 billion in 2016 (Gartner, 2012), the number of PaaS vendors is constantly increasing as well. This development has led to a multitude of PaaS providers, with some specializing in particular technologies (e.g. Java, Ruby, and PHP), some supporting a broad range of technologies (e.g. Heroku[1]), and others allowing almost any composition of technologies by providing a configurable stack (e.g. Cloudfoundry[2]).

However, this large set of providers - with each having their own features, supported technologies, APIs and management approach - causes difficulties when customers have to choose the right vendor for their needs and especially when they need to switch vendors once the service is established and fully functional (Gibson et al. 2012). To reduce the impact of vendor lock-in[3] (Catteddu & Hogben, 2009) (Pearson & Benameur, 2010), in this paper we propose a PaaS-Cloud management application that allows the

[1] https://www.heroku.com
[2] http://www.cloudfoundry.com/
[3] A vendor lock-in occurs, when the configuration of a deployment at a certain PaaS provider varies heavily compared to other providers and thus renders the possibility of a provider switch highly impractical and uneconomic.

deployment of a cloud application at different PaaS providers through a unified API. It enables cloud customers to select their preferred provider, i.e. to easily switch providers concerning their needs and thus avoiding the issue of vendor lock-in. Furthermore, different means of PaaS application management will be discussed and evaluated, resulting in the creation of a proof-of-concept implementation of an application that provides an interface between a selected set of PaaS providers.

The paper is structured as follows. In Section 2 we describe related work with respect to the management of PaaS applications. In Section 3 different PaaS management possibilities are discussed. In Section 4 we present PaaSPort, our proof-of-concept PaaS cloud management implementation that prevents PaaS vendor lock-in through the use of a unified management API. Finally, in Section 5 we draw conclusions.

2. RELATED WORK

In the following related work with respect to the management of PaaS applications is described.

2.1 Cloud Application Management for Platforms (CAMP)

In August 2012 a workgroup, whose members stem from several companies active in the cloud environment (including CloudBees, Oracle, Red Hat and Rackspace), released an initial version 1.0 of a Cloud Application Management for Platforms (CAMP) specification (OASIS, 2014). In 2014, an extended version 1.1 of this specification has been published by OASIS.

This specification aims to provide an API for managing applications in a PaaS environment. Designed to be an open standard, it removes portability issues between different platforms. The general idea of this specification is to provide an API for PaaS systems that masks the underlying infrastructure, such as virtual machines, storage or network. Consequently, the focus of cloud resource access model is switched from low-level resources such as computing power or data storage to applications and their components. Users can access cloud resources and transfer them between providers irrespective to the underlying infrastructure.

The CAMP specification represents cloud applications with a resource-based model. It describes their components, properties (e.g., runtime, configuration, or metadata information) and relationships through a domain-specific language. Finally, CAMP offers a RESTful protocol to change resources or components and thus the state of the application. Although the development of CAMP was started in 2012, the specification is rather new. Hence, its adoption and acceptance is low with respect to cloud PaaS providers. Currently, only a few implementations of the CAMP specification exist, e.g., nCAMP[4] – the proof-of-concept implementation of CAMP 1.1 – or Solum[5], which has just parts of CAMP realized.

2.2 PaaSage Project

The *PaaSage* project is an ongoing effort with the goal to address the findings of the EU Commission Cloud Computing Expert Working Group (Schubert & Jeffery, 2012). The project started in October 2012 under EU FP7-ICT support and is scheduled to be executed until September 2016. The aim of *PaaSage* is to deliver an open and integrated platform to support both design and deployment of cloud applications, together with an accompanying methodology that allows model-based development, configuration, optimization and deployment of existing and new applications, independently of the existing underlying cloud infrastructures (Bubak et al, 2013). As *PaaSage* focuses on model-based development, its approach relies on the definition of the cloud modeling language (CML), which is applied during application configuration and development lifecycles. The further evaluation of modeling requirements at system level enables dynamic application management, including the selection of proper target deployment infrastructure that satisfies criteria defined in the configuration models provided at design-time (Ferry et al, 2013).

Although it promises to deliver valuable results, this project is still in an early execution phase and is expected to complete in 2016. Therefore, its results cannot be assessed and compared at the moment.

[4] http://ec2-107-20-16-71.compute-1.amazonaws.com/campSrv/
[5] https://wiki.openstack.org/wiki/Solum

3. PaaS APPLICATION MANAGEMENT POSSIBILITIES

In this section several methods of managing an application hosted in a PaaS environment are discussed. The conclusion will focus on the optimal choice for realizing the unified PaaS management application, which will further be discussed in Section 4.

3.1 Operating System Shell

Using the operating system (OS) shell as an environment for an application management tool can basically be done in one of the following ways:

Pure Source Code Management (SCM):

With this method, the PaaS provider simply offers one or multiple SCM repositories (e.g. GIT, CVS or SVN) for hosted applications. Whenever new changes are committed to the repository, a new build and update of the hosted application will be triggered on the provider's server. Additional lifecycle management options can be offered in various kinds, e.g. builds are only triggered by pushing on a specific branch.

Pure console application:

A pure console application is most commonly implemented through the use of a scripting language such as Python or Ruby. Such an application will usually communicate with the PaaS provider by using web service calls. It offers commands for managing the application state, scaling options, and deployment.

Hybrid:

This approach combines both of the previous techniques and typically employs a console application. It provides application lifecycle management methods as well as a variety of other configuration options such as scaling, testing and an SCM repository. Several PaaS providers use such tools either as a supplement to their other application management offerings (as described in the following sections) or as a core technology for using their services. Examples are Heroku Tool-belt, OpenShift client tools, or Cloudfoundry CLI.

Advantages of console applications are a concise interface to the functionalities of the PaaS service and the possibility to incorporate them into scripts, which might be useful in more advanced development workflows. On the other hand, console applications might be cumbersome from the point of usability. In comparison to other GUI tools, the tasks like setting up a new application environment (selecting hosting location, application server or other services) could require additional user effort. Furthermore, dependency on various environments, such as operating system and installed scripting language interpreter, could be an issue.

3.2 IDE Extension

This approach pursues the idea of minimizing the amount of separate tools/environments needed for application lifecycle management by integrating the PaaS functionality into commonly used IDEs (integrated development environments) such as Eclipse or Visual Studio. The most basic functionality provided by such plug-ins is creating a project with its settings adjusted for the PaaS environment as well as the deployment to a remote server. The potential extensiveness of a plug-in is on par with that of console applications. In comparison to other management alternatives, the IDE extension approach has advantages when it comes to usability and efficiency, as it avoids frequent switching of the user environment between IDE, browser, and a console. A possible disadvantage is, from a vendor's perspective, higher maintenance effort, since supporting various IDE versions requires adjustments in the plug-in. Real world examples of this approach are e.g. the Heroku Eclipse plug-in, the Cloudbees toolkit for Eclipse, and the Google Web Toolkit plug-in for Eclipse.

3.3 Web Application

Web applications (or web consoles) are the basic application management utility for almost all PaaS providers. They are commonly used to create a basic environment for new applications (e.g. setting up a Java application server) as well as to manage settings for already deployed applications. Many vendors implement application updates not only through the use of console applications, SCM integration, or IDE plug-ins, but also by providing upload sections in their web consoles (e.g. uploading a .war file). Since the appearance of a web application is - in comparison to the previous approaches - highly customizable, they could potentially range from basic command line emulations to fully-fledged IDEs. This approach prevents dependency issues with operating systems or required frameworks and does not require local software installation, and consequently its update and maintenance for various systems. The only compatibility factor is the type of browser used by the client, which might force compromises, as the feature set of browsers of different vendors might diverge considerably. Furthermore, JavaScript code still does not run as fast as a native code. However, its speed is comparable to those of other interpreter languages such as Python or Ruby, and an argument can be made that typical PaaS management tasks are not computationally intensive.

3.4 Discussion

All of the above methods should be adequate for managing applications hosted on a specific PaaS provider. As our unifying PaaS management application should be able to handle multiple applications hosted on various providers, the complexity and set of constraints increase. A clear view on the state of an application across multiple providers and instances can be achieved more easily using a graphical interface, which rules out the console application approach. As there are many different IDEs for all technologies supported by PaaS providers, a unifying PaaS management application either would have to be implemented for the majority of them, or users will be forced to possibly use an IDE solely for this purpose. The web-based approach avoids the aforementioned drawback, while still offering a comparable set of features. Thus, in the scope of this paper, a web-based approach is the optimal choice for a unified PaaS management application.

4. PaaSPort – PaaS MANAGEMENT APPLICATION

The PaaSPort application aims to offer a platform for developers, allowing them to manage and distribute their applications on a variety of PaaS providers. PaaSPort serves as a proof-of-concept implementation for the use-case of hosting an application at different providers and using their APIs for application management. The functionality offered by PaaSPort is accessible both through a web-based interface as well as a REST API. In the following sections we will discuss the selection of supported PaaS providers and the APIs published by them, the architecture of the application itself, as well as its feature set and REST API.

4.1 Requirements and Feature Set

PaaSPort was designed to offer the functionality of application deployment to selected providers in an easy, accessible and extensible manner. To achieve this, the application was built upon the following cornerstones:

User based: A user may login to PaaSPort and register her credentials to a set of PaaS providers in the system. PaaSPort will then use this information to offer deployments to the specified providers.

Manage user applications: A user manages a set of applications that are registered in PaaSPort, defined by their name and a related .war file. The user can create, read, update, and delete (CRUD) applications.

Manage hosted applications: Each application that a user registers in PaaSPort is eligible for deployment to a specified PaaS provider, thus becoming a "hosted application". To do so, the user has to adjust provider specific settings (i.e. framework version of the PaaS stack, allocated memory or hosting location). Afterwards, the state of the hosted application can be observed and altered through PaaSPort (i.e. scale up/down, set allocated memory, or start/stop the application).

Web Console and RESTful API: All of the above functionality is accessible through both a web console interface as well as a RESTful API.

4.2 Architecture

In this section the basic architecture of the PaaSPort management application and its components is described. Figure 1 illustrates this basic architecture.

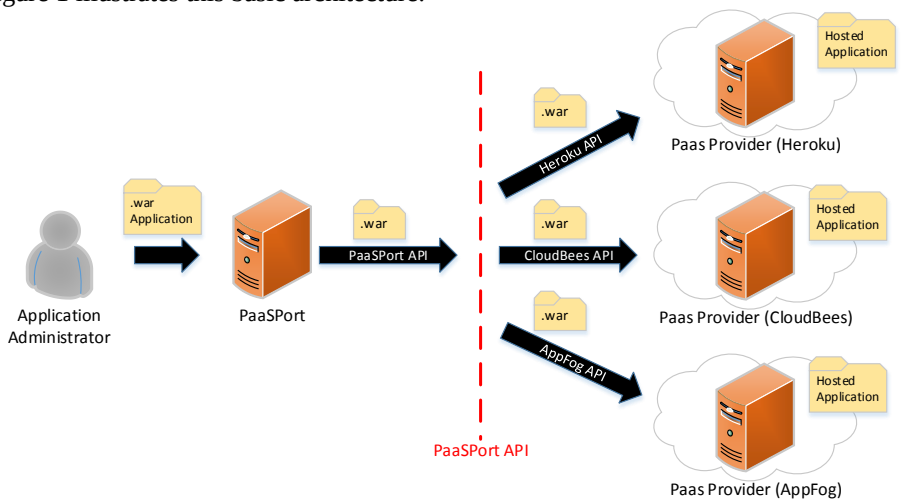

Figure 1 - PaaSPort Architecture

The following components are part of the PaaSPort architecture:

- *Application Administrator (User):* The application administrator (user) is registered with PaaSPort. A user can have multiple applications assigned to administer.
- *PaasPort:* The unified cloud PaaS management application supporting .war application deployment at multiple cloud PaaS providers.
- *PaasPort API:* A unified PaaS API facilitating access to different PaaS provider APIs through a single interface. The PaaSPort API is REST-based.
- *PaaS Provider*: A particular PaaS provider/vendor which can deploy and host .war file applications.
- *.war Application*: An application registered with PaaSPort. Each application has a Java .war file connected to it and is assigned to a specific user.
- *Hosted Application:* A hosted application is an application, which has been deployed to a specific PaaS provider. It is assigned to a .war application and a PaaS provider.

4.3 Implementation

Technically, the main aspect of this implementation is consuming web services of different PaaS providers and exposing their combined functionality through a single web service. Although many providers offer similar functionalities, the manner, in which their services are accessed, often varies. The web services offered by the PaaS providers are often the backbone for their other management tools, since they act as a central access point for all of the providers' services. For example, all CLI (command line interface) tools offered by various providers function as an easy-to-use interface for accessing the providers' web service.

For the prototypical implementation and illustration of PasSPort applicability, three PaaS providers were selected. The main criterion for the selection of a provider was the possibility to deploy an application easily by uploading a Java .war file. Thus, the following providers were selected: Heroku, CloudBees, and AppFog. Heroku offers a REST API as well as the possibility of deploying an application by uploading it as a .war file. CloudBees offers an extensive Java library that encapsulates the functionality of its web API. AppFog offers a REST API covering all basic actions for managing the application and its services.

The web console of PaaSPort was implemented using JSF[6]. Each web page has a JSF bean assigned to it, providing all necessary data and operations. The JSF beans communicate with the data access layer, which in

[6] Java Server Faces, http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

turn accesses a Derby database[7]. This database stores relevant information for the following architectural components: Application Administrator (User), .war Application, Hosted Application, and PaaS Provider.

The RESTful PaaSPort API was implemented using the Jersey library[8], designed as a separate layer in the application that communicates with the data access layer. The data format for all information transmitted by the web service is JSON. The RESTful API of PaaSPort is designed to expose the data components of interest. Table 1 shows details on the RESTful PaaSPort API.

Table 1 - RESTful PaaSPort API

URL endpoint	HTTP Method	Description
/apps	GET	Returns a list of applications registered by the user.
/app/<appname>	POST	Creates a new application.
/app/<appid>	DELETE	Deletes an application.
/app/<appid>	GET	Retrieves information about a specific application.
/app/<appid>/ <hostedappname>	POST	Deploys an application to a provider and thus creates a new hosted application.
/app/<appid>/<hostedappid>	PUT	Updates settings for a hosted application.
/app/<appid>/<hostedappid>	DELETE	Undeploys and deletes a hosted application.

In order to secure the API and provide a means of authentication, HTTP Basic Authentication has been implemented. Any user accessing the API has to set the proper HTTP header, thus providing her credentials in every request. This simple mechanism has been implemented to provide a basic access control. However, more advanced authentication mechanisms (e.g., two-factor based) can be easily integrated.

4.4 Workflow

This section will demonstrate some of the actions that can be performed using the PaaSPort web console. It is assumed that the user is already registered with the system and authenticated for this session.

Error! Reference source not found. depicts all currently registered .war Applications. The associated file can be downloaded by clicking on it. The button "Create New App" redirects to the Add Application page (**Error! Reference source not found.**). By clicking the name of the Application, the user is redirected to the Hosted Application overview (Figure 6).

In **Error! Reference source not found.** a new Application with the name "Test Application" is created. Required fields are the name of the Application and the associated .war file. Upon completing the creation, the user is redirected back to the Application overview (**Error! Reference source not found.**).

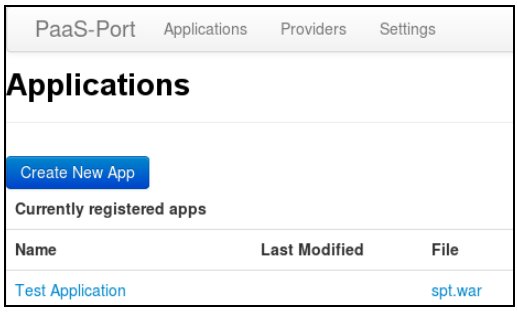

Figure 2 – PaasPort .war Application Overview

Figure 3 - Adding a .war Application to PaaSPort

Error! Reference source not found. depicts the Hosted Application overview. In this screenshot, the Application "Test Application" has not yet been deployed to any PaaS Provider. To do so, the user has to select a provider from the dropdown-menu and click the button "Deploy to provider". This will redirect the user to the "Deploy to Provider" page (**Error! Reference source not found.**).

[7] http://db.apache.org/derby/

[8] https://jersey.java.net/

If the user chooses to deploy to Heroku, she will be redirected to this provider specific deploy page (**Error! Reference source not found.**). Required fields for this provider are the name of the Hosted Application, the region for the infrastructure to which the Application will be deployed and the Stack (the setup of the underlying system). Upon completing the deploy operation, the user will be redirected back to the Hosted Application overview (Figure 6), which now lists the newly created Hosted Application.

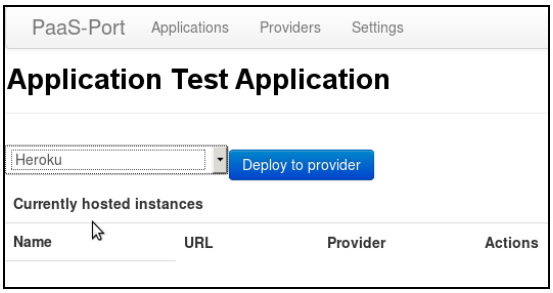

Figure 4 - Choosing the PaaS Provider to which the .war application will be deployed

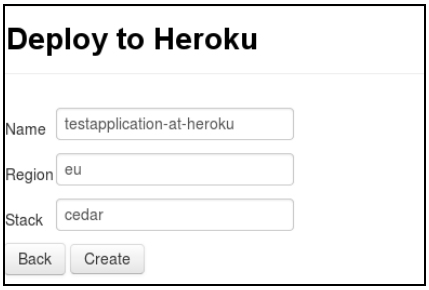

Figure 5 - Providing provider-specific deployment information

By clicking the name of the Hosted Application, the user will be redirected to a provider specific settings page (Figure 7). The URL field links to the public page of the now deployed and running Application. By pressing the "Delete" button, the Hosted Application will be deleted both at the PaaS provider it was deployed to and in PaaSPort.

The provider specific Hosted Application settings page (Figure 7) offers information about the current state of the deployed Application. Additionally, it is also possible to change provider specific settings. In the case of Heroku, the user is able to increase or decrease the number of currently allocated Dynos (Heroku-defined container).

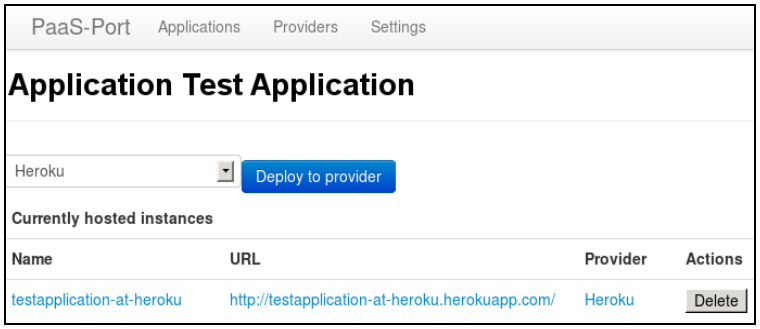

Figure 6 - Overview of all Hosted Application instances

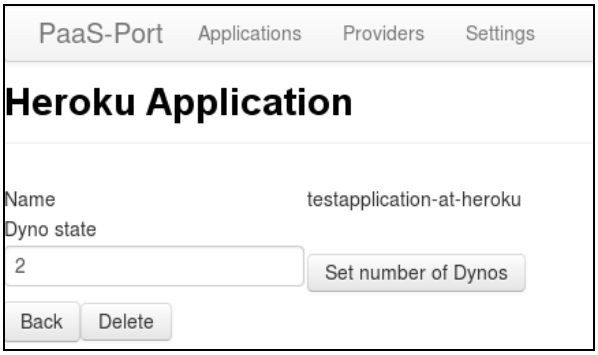

Figure 7 - Monitoring and updating a Hosted Application

5. CONCLUSIONS

As the PaaS landscape has been steadily growing for the last few years, more and more providers emerge. Meanwhile, the existing providers and early adopters are constantly upgrading and refining their products. PaaS has established itself for a wide variety of customers, from companies, often using private PaaS as an easily manageable and scaling infrastructure, to independent developers, who get a competitive application environment with minimal configuration overhead.

However, by comparing some of the predominant PaaS providers, it becomes apparent that there are substantial differences in many aspects of the offered functionality. Each provider caters to slightly different needs and thus finding the right PaaS provider requires extensive research and analysis. Furthermore, the task of changing PaaS providers proves to be difficult, as there is no common implemented standard for cloud environments, and hence a provider switch might be impractical or uneconomic (vendor lock-in effect).

As a proof-of-concept implementation effort, PaaSPort was created with the aim to eliminate vendor lock-in by providing an abstracted and consolidated management interface. PaaSPort aims furthermore to enable users to manage uploaded applications transparently and efficiently using a unified web interface. PaaSPort allows users to upload a Java web application, packed as a .war file, and deploy and manage that application on AppFog, Cloudbees, and Heroku. Morevover, all functionality is exposed through a REST web API, which standardizes operations that are common across all providers, while leaving room for provider-specific operations. PaaSPort therefore functions as a single interface for the basic management of applications on three different PaaS providers.

PaaSPort currently demonstrates the support for three different PaaS providers. Support for additional providers could be implemented, if they expose their functionality through a web API and accept .war file uploads. Furthermore, the application can be extended to support SCM systems, such as GIT, to not only rely on file upload for application deployment.

REFERENCES

Bubak, Marian, et al, 2013. PaaSage: Model-Based Cloud Platform Upperware. *Demonstration & Exhibition at the Future Internet Assembly*

Catteddu, D. and Hogben, G., 2009. *Cloud Computing - Benefits, risks and recommendations for information security*, ENISA report

Chieu, T.C., Mohindra, A., Karve, A. A., Segal, A., 2009. Dynamic scaling of web applications in a virtualized cloud computing environment. *In IEEE International Conference on e-Business Engineering (ICEBE'09)*. pp. 281-286.

Ferry, N., Rossini, A., Chauvel, F., Morin, B., Solberg, A., 2013. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. *In CLOUD 2013: IEEE 6th International Conference on Cloud Computing*. pp. 887-894.

Gartner, 2012. *Gartner says worldwide platform as a service revenue is on pace to reach $1.2 billion*. http://www.gartner.com/newsroom/id/2242415.

Gibson, J., Rondeau, R., Eveleigh, D., and Tan, Q., 2012. Benefits and challenges of three cloud computing service models. *In Fourth International Conference on Computational Aspects of Social Networks (CASoN)*. pp. 198-205.

Lawton, G., 2008. Developing software online with platform-as-a-service technology. *In Computer*. Vol. 41, No. 6, pp. 13-15.

OASIS, 2014, *Cloud Application Management for Platforms Version 1.1*, http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf

Pearson, S. and Benameur, A., 2010. Privacy, Security and Trust Issues Arising from Cloud Computing. *In IEEE Second International Conference on Cloud Computing Technology and Science*. pp. 693–702

Schubert, Lutz; Jeffery, Keith. Advances in clouds. *Report of the Cloud Computing Expert Working Group. European Commission*, 2012.