Journal of Computer and Communications, 2014, 2, 1-7
Published Online October 2014 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2014.212001

Scientific Research

Lightweight Virtualization Cluster How to Overcome Cloud Vendor Lock-In

Nane Kratzke

Department for Electrical Engineering and Computer Science, Center of Excellence CoSA, Lübeck University of Applied Sciences, Lübeck, Germany
Email: nane.kratzke@fh-luebeck.de

Received 1 July 2014; revised 4 August 2014; accepted 26 August 2014

Abstract

To overcome vendor lock-in obstacles in public cloud computing, the capability to define transferable cloud-based services is crucial but has not yet been solved satisfactorily. This is especially true for small and medium sized enterprises being typically not able to operate a vast staff of cloud service and IT experts. Actual state of the art cloud service design does not systematically deal with how to define, deploy and operate cross-platform capable cloud services. This is mainly due to inherent complexity of the field and differences in details between a plenty of existing public and private cloud infrastructures. One way to handle this complexity is to restrict cloud service design to a common subset of commodity features provided by existing public and private cloud infrastructures. Nevertheless these restrictions raise new service design questions and have to be answered in ongoing research in a pragmatic manner regarding the limited IT-operation capabilities of small and medium sized enterprises. By simplifying and harmonizing the use of cloud infrastructures using lightweight virtualization approaches, the transfer of cloud deployments between a variety of cloud service providers will become possible. This article will discuss several aspects like high availability, secure communication, elastic service design, transferability of services and formal descriptions of service deployments which have to be addressed and are investigated by our ongoing research.

Keywords

Cloud Computing, Vendor Lock-In, Cross-Platform, Lightweight Virtualization, Container

1. Introduction

Cloud computing can be seen as a new IT service delivery paradigm often called a “programmable data center”

How to cite this paper: Kratzke, N. (2014) Lightweight Virtualization Cluster How to Overcome Cloud Vendor Lock-In. *Journal of Computer and Communications*, **2**, 1-7. http://dx.doi.org/10.4236/jcc.2014.212001

[1]. These web accessible "programmable datacenters" provide several interesting features. They allow scaling up and down resource usage on an as-needed basis likely to provide cost advantages (especially in peak load scenarios [2] [3]) due to their economical pay-as-you-go principle and technical quick scaling capabilities. Economically they turn fixed costs into variable costs making theme specially interesting for innovative start-up companies not able to do significant up-front investments. While start-up companies in most cases do not have a well established business model, small and medium sized enterprises have. Therefore small and medium sized enterprises are often innovative on the one hand but also very careful in calculating risks on the other hand. Beside chances these companies are also focused to avoid hardly calculable risks.

So beside the above mentioned advantages, common security, governance, availability, cost concerns and especially technological vendor lock-in worries often come along with cloud computing which is especially true for small and medium-sized enterprises.

While security and governance concerns often can be answered by encryption [2], and cost concerns can be answered by cost-based decision making models [2] [3], vendor lock-in problems stay. From a technological point of view actual state of the art cloud-based software, services are often characterized by a highly implicit technological dependency on underlying hosting cloud infrastructures. This article focuses on this vendor lock-in problem and derives several aspects to be solved to overcome vendor lock-in problems making cloud computing a better fit for startups, for small, medium and big sized enterprises as well.

Vendor lock-in is defined to make a customer dependent on a vendor for products and services, unable to use another vendor without substantial switching costs. This is exactly the actual state of the art of IaaS (infrastructure as a service) cloud computing. On one hand IaaS provides a commodity service. A commodity describes a class of demanded goods (in case of IaaS computing, storage and networking services) without qualitative differentiation. So the market treats instances of the goods as (nearly) equivalent. Or simpler: a virtual server provided by Google Compute Engine (GCE) is not better than a server provided by Amazon Web Services (AWS). Both can be used to host a customer specific service. So IaaS provides commodities from a simple point of view.

On the other hand reality looks more complex. Although a lot of cloud computing services are commodity services, a switch from one service provider to another service provider is in most cases not as easy as buying coffee from a different supplier. This is mainly due to inherent dependencies on underlying cloud infrastructures. These dependencies are often subliminal generated by cloud service provider's specific (non-standardized) service APIs.

To overcome subliminal generated vendor lock-ins, this paper proposes a concept called *lightweight virtualization cluster* (LVC) relying on operating system virtualization. Lightweight (or operating system) virtualization is gaining more and more attention in cloud computing.

The remainder of this paper is structured as follows. Section 2 shows by example how inherent infrastructure dependencies are generated and what strategies exist to avoid this. Nevertheless every identified strategy has some shortcomings so far. Section 3 advocates a lightweight virtualization approach and derives in Section 4 a lightweight virtualization cluster concept intentionally guiding our ongoing research. Section 5 summarizes the key points how to overcome vendor lock-in in cloud computing, how this could be used to converge existing cloud technologies into something which is called Meta Cloud by Dustdar *et al*. [4] and provides an outlook for our planned research.

Please remember, the intent of this paper is not to provide concrete research results. The intent of this paper is to structure our research and foster discussions.

2. How Vendor Lock-In Emerges Technically in Cloud Computing

It is easy to make cloud computing vendor lock-in problems obvious by looking at an example. Taking a look on provided services (see **Table 1**) by Amazon Web Services (AWS) we see that AWS provides over 30 services but only nine of them can be rated as commodity (own rating). Using cloud computing means it is likely that (horizontal) scalable and elastic systems will be deployed. That is why cloud computing is used in most cases. So in most cases scalable distributed systems will be deployed to cloud infrastructures. Components of distributed systems need to interact and this can be tricky in details for system engineers (especially with scalable system designs). While the core components (virtualized server instances) of these distributed systems can be deployed using commodity services (that is in case of AWS mainly done using the EC2 service) all further services to integrate these virtualized server instances in an elastic and scalable manner are provided by non-

Table 1. AWS provided cloud computing services (according to whitepaper overview of Amazon webservices [5]).

AWS Service Category	Service	Category	Commodity (Own Rating)	LVC (Relevant)
Database	DynamoDB	Scalable NoSQL Data Store	No	
	Elastic Cache	In-Memory Cache	Yes	
	RDS	Relational Database	Yes	
	Red Shift	Petabyte-Scale Data Warehouse	Yes	
Storage & CDN	S3	Scalable Storage	No	+
	EBS	Network Attached Block Device	Yes	
	Cloud Front	Content Delivery Network	No	
	Glacier	Archive Storage	No	
	Storage Gateway	Integrates On-Premises IT with Cloud Storage	No	
	Import Export	Ship Large Datasets	No	
Analytics	Elastic MapReduce	Managed Hadoop Framework	Yes	
	Kinesis	Real-Time Data Stream Processing	No	
	Data Pipeline	Orchestration for Data-Driven Workflows	No	
Compute & Networking	EC2	Virtual Servers	Yes	+
	VPC	Virtual Private Network	No	+
	ELB	Elastic Load Balancing	No	+
	Work Spaces	Virtual Desktops	No	
	Auto Scaling	Automatically Scale Up and Down	No	+
	Direct Connect	Dedicated Network Connection to AWS	No	
	Route 53	Scalable Domain Name System	Yes	
Deployment & Management	Cloud Formation	Templated AWS Resource Creation	No	
	Cloud Watch	Resource and Application Monitoring	No	
	Elastic Beanstalk	AWS Application Container	No	
	IAM	Identity and Access Management (Secure Access Control)	No	
	Cloud Trail	User Activity Logging	No	
	Ops Works	DevOps Application Management Service	Yes	
	Cloud HSM	Hardware-Based Key Storage for Compliance	No	
App Service	Cloud Search	Managed Search Service	Yes	
	Elastic Transcoder	Scalable Media Transcoding	No	
	SES	Email Sending Service	No	
	SNS	Push Notification Service	No	
	SQS	Message Queue Service	No	
	SWF	Workflow Service for Coordinating App Components	No	
	App Stream	Low-Latency Application Streaming	No	

commodity services. In case of AWS it is very likely that services like elastic load balancing (ELB), auto scaling, message queue system (SQS) are needed to design an elastic and scalable system on a AWS infrastructure. But especially these services are non-commodity services binding a deployment to vendor specific infrastructure (in case of our example to AWS).

The same is true for other cloud service providers (but only with a smaller set of provided services). All cloud service providers try to stimulate cloud customers to use non-commodity convenience services in order to bind them to their infrastructure.

From a cloud service provider point of view the wish to bind customers is not astonishing and so the mentioned strategy is more than obvious. Nevertheless from a cloud customer point of view this vendor lock-in should be avoided to remain flexible in operation (e.g. in case a cloud service provider becomes insolvent and has to giving up business). Reflecting actual research literature the following strategies can be identified to overcome vendor lock-in.

- **Industrial top down standardization approaches** like CIMI (Cloud Infrastructure Management Interface) try to define a standardized API to different cloud service providers. So if it is possible to deploy to a provider A via a standardized API it is also possible to deploy to a (standard compliant) provider B. Looking at CIMI [6] in detail will show that typical standardization problems arise like the least common denominator problem. A standardization seems only to work for commodity services (compare **Table 1**) but not for non commodity services necessary for deployments of typical complexity (load balanced, elastic and distributed). Furthermore companies like AWS are not known to be very standard oriented ("premature standardization considered harmful" is a famous statement from Werner Vogels, CTO of Amazon).
- **Open source bottom up approaches** try to do the same like the industrial standardization approach but applying a bottom up strategy. E.g. fog.io [7] is a Ruby-based API to access over 30 cloud service providers. But like CIMI fog.io must use a least common denominator approach. More complex services like load balancing, auto scaling, messaging, etc. are not accessible (or if, they are in fact provider specific). A main problem of the applaudable fog.io library is its very provider specific parameter sets. Companies like Netflix or Paypal develop open source suites like Asgard[1] or Aurora[2] making more complex services for load balancing, auto scaling, etc. accessible for private cloud infrastructures like Eucalyptus[3] or OpenStack[4]. But a transfer between these worlds is hardly possible right now because they seem to rely on a lot of very infrastructure specific features.
- More formal **research approaches** try to formalize the deployment description by defining mostly XML-based deployment languages (compare [8]-[11]). Using a deploying component capable to interpret the deployment language theses approaches are capable to deploy these formal descriptions to the APIs of several service providers (even into federated clouds). But these deployed systems are not elastic (so do not provide automatic up and down scaling). Furthermore they do not face the problem how to move a deployed distributed system from one service provider to another.

So far all of the strategies seem to have shortcomings especially in handling dynamic cloud deployments of typical complexity (so deployments which are elastic and load balanced).

3. Avoid Vendor Lock-In Using a Operating System Virtualization Approach

IaaS cloud computing is heavily affected by hypervisor virtualization. Multiple virtual servers providing different operating systems can be run on one physical host system. So customers can run the operating system they want. Upcoming lightweight virtualization technologies like Docker [12] seem auspicious but rely on operating system virtualization. Maybe that is why operation system virtualization is not used for avoiding vendor lock-in so far. Using lightweight (operating system) virtualization the virtualized operating system has to be the same like the hosting operating system. So lightweight virtualization is limited on the one hand but one the other hand generates a better hosting density. In general it is possible to host more lightweight virtualizations on a physical host than with hypervisor based virtualizations. Especially unixoid systems support this form of light-weight virtualization (see **Table 2**).

[1] https://github.com/Netflix/asgard/wiki (last access 1st July 2014).
[2] https://github.com/paypal/aurora (last access 1st July 2014).
[3] https://www.eucalyptus.com (last access 1st July 2014).
[4] http://www.openstack.org (last access 1st July 2014).

Table 2. Lightweight virtualization mechanisms on various operating systems.

Operating System	Virtualization Mechanism	Link for Further Information (last access 30th June 2014):
Linux	Docker	http://www.docker.io
	LXC	http://linuxcontainers.org
	OpenVZ	http://openvz.org
Solaris	Solaris Zones	http://docs.oracle.com/cd/E26502_01/html/E29024/toc.html
FreeBSD	FreeBSD Jails	http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html
AIX	AIX Workload Partitions	http://www.ibm.com/developerworks/aix/library/au-workload/index.html
HP-UX	HP-UX Containers (SRP)	https://h20392.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=HP-UX-SRP
Windows	Sandboxie	http://www.sandboxie.com

Operating system virtualization can be deployed to any public or private cloud service provider due to its lightweightness. Operating system virtualization can be frictionless applied on top of hypervisor virtualization and is therefore deployable to the commodity part of any cloud service infrastructure. Recently the open source tool Docker [12] made this approach very popular in the DevOps community.

"Like a virtual machine, a Docker container can package an application into a single file, freeing the developer from worrying about the underlying system software. Unlike full virtual machines, though, a Docker container does not include a full OS, but rather shares the OS of its host—in Docker's case, Linux.

As a result, Docker containers can be faster and less resource-heavy than virtual machines. A full virtual machine may take several minutes to create and launch, whereas a container can be initiated in seconds. Containers also offer superior performance for the applications they contain, compared to running the application within a virtual machine, which incurs the overhead of running through a hypervisor.

[…]

The level of ecosystem support Docker has gained is stunning and it speaks to the need for this kind of technology in the [*cloud*] *market and the value it provides.*

(Joab Jackson, cio.com[5])

So Docker is a lightweight virtualization tool that can package an application and its dependencies in a virtual container that can run on any Linux server. A containerized application can run on premise, on public cloud, on private cloud oron bare metal. And these containers can be transferred from one Docker host to another providing therefore transferability out of the box. From a conceptual point of view a containerized application can be seen as a deployable service runnable on a Docker host. Several Docker hosts could build a lightweight virtualization cluster. Due to the fact that Docker itself relies on operating system virtualization the Docker infrastructure can be deployed to the commodity part of any IaaS cloud service provider. Operating system virtualization is proofed to work frictionless on top of hypervisor based virtualization.

So lightweight virtualization is a perfect abstraction layer for the commodity part of any IaaS cloud service infrastructure. Unlike industrial top down or open source bottom up standardization approaches lightweight virtualization defines an abstraction layer-out of the box only depending on commodity services (virtual servers). On the other hand this approach is restricted to operating system virtualization and therefore only applicable to unixoid operating systems from a pragmatic point of view. Nevertheless according to common server operating market shares Unix based servers are used by almost 2/3 of all public accessible websites[6]. So the here presented approach is likely to be applicable to 2/3 of the relevant problem domain which is good from authors point of view.

4. Requirements of a Lightweight Virtualization Cluster

The lightweight virtualization abstraction layer must provide several features normally provided in a non-com-

[5]http://www.cio.com/article/2375628/virtualization/docker-all-geared-up-for-the-enterprise.html (last access, 9th September 2014).

[6]W3techs.com footnote 6, http://w3techs.com/technologies/details/os-unix/all/all (latest access 30th June 2014).

modity way by cloud service providers (especially auto scaling, load balancing) but not provided by lightweight virtualization tools like Docker [12] out of the box. For simplicity this abstraction layer is called a LVC (lightweight virtualization cluster). A LVC has the job to host services, enable (load balanced and secure) communication between services, up and down scale services to loads, extract and inject persistent service states. A LVC is a cluster and therefore consist of several hosts capable to host single services or complex service deployments (services consisting of other services). And furthermore a LVC must be deployable to any IaaS public or private cloud service or bare metal infrastructure.

In ongoing research we want to define necessary concepts of such a LVC. LVCs should support the following initial features which can be derived from actual strategies (see Section 2) to overcome vendor lock-in obstacles present in cloud computing and addresses common security, governance, and availability worries (see Section 1).

1) High available deployment of services to a LVC;
2) Secure communication between services in a LVC;
3) Convenient auto scalability features of a LVC;
4) Convenient load balancing features of a LVC;
5) Distributed and scalable service discovery and orchestration within a LVC;
6) Transfer of service deployments between LVCs;
7) Formal description of service deployments deployable to a LVC.

We furthermore want to study how transferable services for this approach have to be designed from a distributed systems software engineering point of view. Especially the state of services (e.g. a database) must be handled adequately while transferring deployments from one virtualization cluster to another. Services could be stateless (which is perfect from a distributed point of view) but also have an ephemeral service state or even a persistent (and often distributed) service state (which is likely the most tricky aspect to handle, imagine a distributed database being transferred from one cluster to another).

5. Conclusions and Outlook

Small and medium sized enterprises have established business models and resulting incomes. Other than start-up companies these enterprises share common security, availability, cost concerns and especially technological vendor lock-in worries about cloud computing (see Section 1). Although these worries can be answered by encryption, by appropriate availability statistics, and by available and appropriate cost estimation models [2] [3] [13] technological vendor lock-in problems stay due to a mixture of a lack of standardization, a lack of open source tools with cross provider support or shortcomings of actual cloud deployment languages (see Section 2). And although operating system virtualization is not as flexible as hypervisor-based virtualization, it provides a “natural” and immanent cloud infrastructure abstraction layer. Recent popularity of lightweight virtualization tools (e.g. Docker) shows the increasing interest of operating system virtualization to cloud computing (see Section 3).

This paper proposed a lightweight virtualization abstraction layer which can be deployed to any IaaS cloud infrastructure to overcome cloud vendor lock-in. Furthermore this paper derived several core requirements of such an abstraction layer and aligned these requirements with ongoing research questions and conceptual feature requirements of a lightweight virtualization cluster.

By defining concrete components for a research concept, this paper titled lightweight virtualization cluster (LVC) current vendor lock-in problems in cloud computing (IaaS) could be reduced. This would help especially small and medium sized enterprise in adopting cloud strategies. Our ongoing research contributes to several postulations made in [3] postulating that all necessary technologies for a meta-cloud are already present but not integrated in an appropriate manner. Authors think that a lightweight virtualization cluster could contribute substantially to overcoming vendor lock-in problems.

Acknowledgements

Research project Cloud TRANSIT is funded by German Federal Ministry of Education and Research. Let me thank Lübeck University (Institute for Telematics) and fat IT solution GmbH for accompanying and supporting the project Cloud TRANSIT.

References

[1] Barr, J. (2010) Host Your Web Site in the Cloud: Amazon Web Services Mad Easy. Sitepoint.

[2] Kratzke, N. (2012) Cloud Computing Costs and Benefits—An IT Management Point of View. In: Ivanov, I., van Sinderen, M. and Shiskov, B., Eds., *Cloud Computing and Services Sciences*, Springer, New York, 185-203.

[3] Kratzke, N. (2011) Overcoming Ex Ante Cost Intransparency of Clouds—Using System Analogies and a Corresponding Cost Estimation Model. *CLOSER* 2011—1*st International Conference on Cloud Computing and Services Science* (Special Session on Business Systems and Aligned IT Services—BITS 2011), Noordwijkerhout, 7-9 May 2011, 707-716.

[4] Satzger, B., Hummer, W., Inzinger, C., Leitner, P. and Dustdar, S. (2013) Winds of Change: From Vendor Lock-In to the Meta Cloud. *Internet Computing*, *IEEE*, **17**, 69-73.

[5] Amazon Web Services (2014) Whitepaper "Overview of Amazon Web Services". https://s3.amazonaws.com/awsmedia/AWS_Overview.pdf

[6] DMTF, CIMI (Cloud Infrastructure Management Interface) (2014) Last Access 30th June 2014. http://dmtf.org/standards/cloud

[7] Fog.io (The Ruby Cloud Services Library) (2014) Last Access 30th June 2014. http://fog.io

[8] de Alfonso, C., Caballer, M., Alvarruiz, F., Molto, G. and Hernandez, V. (2011) Infrastructure Deployment over the Cloud. *Proceedings of* 2011 *IEEE* 3*rd International Conference on Cloud Computing Technology and Science* (*CloudCom*), Athens, 29 November-1 December 2011, 517-521.

[9] Juve, G. and Deelman, E. (2011) Automating Application Deployment in Infrastructure Clouds. *Proceedings of* 2011 *IEEE* 3*rd International Conference on Cloud Computing Technology and Science* (*CloudCom*), Athens, 29 November-1 December 2011, 658-665.

[10] Lenk, A., Danschel, C., Klems, M., Bermbach, D. and Kurze, T. (2011) Requirements for an IaaS Deployment Language in Federated Clouds. *Proceedings of SOCA*, Irvine, 12-14 December 2011, 1-4.

[11] Murphy, S., Gallant, S., Gaughan, C. and Diego, M. (2012) U.S. Army Modeling and Simulation Executable Architecture Deployment Cloud Virtualization Strategy. 12*th IEEE/ACM International Symposium on Cluster*, *Cloud and Grid Computing* (*CCGrid*), Ottawa, 13-16 May 2012, 880-885.

[12] Merkel, D. (2014) Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*, **2014**, No. 2.

[13] Talukader, A.K., Zimmermann, L. and Prahalad, H. (2010) Cloud Economics: Principles, Costs and Benefits. *Cloud Computing—Computer Communications and Networks*, **4**, 343-360.

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.