The Docker Ecosystem Needs Consolidation

René Peinl and Florian Holzschuher
Institute of Information Systems, Hof University, Alfons-Goppel-Platz 1, Hof, Germany
{rene.peinl, florian.holzschuher2}@iisys.de

Keywords: Cloud Computing, Management Tools, Micro-services, System Integration, Docker, Container.

Abstract: Docker provides a good basis to run composite applications in the cloud, especially if those are not cloud-aware, or cloud-native. However, Docker concentrates on managing containers on one host, but SaaS providers need a container management solution for multiple hosts. Therefore, a number of tools emerged that claim to solve the problem. This paper classifies the solutions, maps them to requirements from a case study and identifies gaps and integration requirements. We conclude that the Docker ecosystem could help moving from IaaS and PaaS solutions towards a runtime environment for SaaS applications, but needs consolidation.

1 INTRODUCTION

Although lightweight operating system (OS) virtualization techniques like Solaris Zones and OpenVZ are long established, it was the release of Docker in March 2013 (Rosen, 2014) that led to mass adoption and even a hype around containerization (Kratzke, 2014). Docker aims at making container technologies easy to use and among other things encourages a service-oriented architecture and especially the micro-service architecture style (Turnbull, 2014). Containers impose less overhead than machine virtualization but still provide less isolation (Scheepers, 2014). In a Software as a Service scenario (SaaS), you therefore cannot guarantee that activities of one customer won't negatively affect other customers, if you are using containers only.

The need for a kind of "application package format" as a basis for composite SaaS offerings following the SOA principles (service-oriented architecture (Papazoglou, 2003)) was already discussed in (Mietzner et al., 2008). Originating from a Platform as a Service (PaaS) use case, Docker should be a good basis to handle the components of a composite application offered in the cloud. It provides an easy and convenient way to create, deploy and configure containers (Rosen, 2014), incl. links to dependent containers on the same host. Micro-services can be briefly summarized as a "loosely coupled service oriented architecture with bounded contexts" (Cockcroft, 2014), where loosely coupled denotes that each service should be independently deployable and bounded contexts means that the service does not have to know anything about its surroundings, but can discover them on its own (cf. Evans, 2003). Enterprise applications are typically complex composite applications, which consist of multiple individual components (Binz et al., 2014) and therefore match micro-services. However, the Docker tools soon reach their limits when it comes to managing containers in a cluster or creating links across multiple hosts (Kratzke, 2014). To overcome those, a myriad of tools is currently in development. We found over 60 tools in "The Docker Book" (Turnbull, 2014), a special issue of the German "developer magazine" dedicated to Docker (Roßbach, 2014) and the "Docker ecosystem" mindmap ("Docker Ecosystem Mindmap", n.d.) with relevance for building an automated Docker cluster solution similar to OpenStack on the IaaS level (Peinl, 2015 for a full list). Docker Inc even counts 50,000 third-party projects on GitHub that are using Docker (Docker, Inc., 2014). Our hypothesis is, that despite the benefits of competition, the time has come to work together on a common cluster project similar to OpenStack to form a comprehensive integrated solution and stable interfaces for the required components in order to make them interchangeable, instead of building yet another tool that solves parts of the challenges, but not all of them and is not integrated with others.

Our methodology was guided by (Chauhan and Babar, 2011), so the rest of the paper is structured as follows. We briefly describe our SaaS project that

serves as a case study to derive requirements. We continue listing and explaining the requirements and then compare them to the functionality of existing tools. We categorize those tools and elaborate on consistent definitions for those categories. We conclude with remaining challenges and an outlook.

2 CASE DESCRIPTION

The goal of the SCHub project (Social Collaboration Hub, funded by the BMBF as part of the FHprofUnt funding, https://www.sc-hub.de) is to develop a distribution-like collaboration solution based on open source software (OSS) that provides end-users with a consistent experience across all systems while using a modular micro-service approach (Cockcroft, 2014). It therefore represents a composite application (Coffey et al., 2010). The solution will be available as Software as a Service (SaaS) in the cloud as well as on premise installation. In order to do that, a number of well-known OSS systems have to be migrated to the cloud and Docker is an obvious choice for supporting that. Since not all systems are capable of handling multiple tenants and customization possibilities are better that way, SCHub uses individual instances of all frontend systems (portal, groupware, …) per tenant and only shares backend systems across tenants (database, mail server, …). Each instance is packaged into a Docker container. To guarantee isolation between instances of different tenants, virtual machines (VM) are used additionally. The VM becomes the Docker host in this case. OpenStack serves as the basis (Sefraoui et al., 2012). Initially, there is only one VM per tenant. When resource limits of this VM are reached, additional VMs are allocated and some containers are migrated to a new host. Storage is provided by Ceph, a software defined storage solution (Koukis, 2013) as either block-level or object storage, depending on the requirements of the service.

Since off-the-shelf systems are used that are integrated with our add-ons, we wanted to change those systems as little as possible in order to stay upwards compatible and benefit from future releases. Therefore, the usage of a PaaS platform was not feasible as it would require adapting the systems to that platform. However, many components of a PaaS solution are still needed, e.g. a load balancer, a central authentication system, database as a service and so on. It turned out, that a new category of cloud offering would be ideal for this case, a kind of runtime environment for SaaS applications (RaaS). Where PaaS targets developers, RaaS targets application administrators. The following chapter lists the requirements for such a solution.

3 REQUIREMENTS

From a provider's perspective, automating the management of the offered services is of vital importance, because management and operation of IT is one of the biggest cost factors today (Binz et al., 2014). Many of the required features are simply a transfer of IaaS management features to Docker. There should be a central list of containers (r1) with an overview of resource usage, IP address, open ports, dependencies and so on. You need a detail view of a container (r2) including a way to change the configuration using the Web UI concerning networking, storage and dependencies. Since the application is built from multiple services and therefore containers, it would be helpful to be able to centrally define a kind of blueprint (r3) that includes all the dependencies and to instantiate the whole solution instead of single containers (r4).

For doing so, the management solution should monitor resource usage of hosts (r5) and automatically choose one with free resources, based on an editable placement strategy (r6). Monitoring should include CPU, RAM, storage and networking, as well as application health. You should be able to configure thresholds so that high CPU utilization over a specified timeframe or low available memory trigger an alert (r7) which in turn can trigger an action like migrating a container to another host. Migration (r8) could be performed by stopping the container, unmounting the storage, starting an identical container on a new host, updating service references (r8b) and mounting the storage (r9) there. Besides storage, there should also be an easy way to pass configuration data to the application inside the container (r10). This data has to be stored in a distributed key/value store (r8a).

For communication between containers across hosts (r11), you ideally need an overlay network or a software defined network (SDN) (Jain and Paul, 2013). Its configuration should be accessible directly from the Docker management UI, e.g., for defining IP address ranges (r12). The Web UI of the SDN could be simply integrated. You need routing of external requests with URLs to tenant-specific container IPs (r13). This routing should include load balancing if multiple container instances are available (r14). There should be a way to review the list of available images (r15) including versions and ideally an association to the containers running that image. If an image is

updated, the admin should be able to trigger a mechanism that propagates the updates to the running instances (r16), e.g. analogous to the migration described above. There should be a way to access the container's console or open an SSH shell respectively (r17) and review the log files (r18), both using the Web UI.

It would also be desirable to have an integration of the underlying IaaS solution, so that you can create new hosts (VMs) from within the Docker Web UI (r19). Finally, there should be an integration with a tenant / customer management solution (r20) where both administrators and customers can review information like the list of Docker containers per tenant, the resulting resource usage, the number of total and monthly active users as well as respective billing information. It could be further argued, that an authentication solution is needed, but we are skipping this requirement, since Docker itself currently has no working mechanism for that anyway.

4 EXISTING SOLUTIONS

We've concentrated our analysis on open source components, although there are a few impressive commercial tools available like StackEngine. The descriptions of the tools capabilities are based on the projects' websites. We have installed and tested only the most promising systems.

4.1 Host Operating System

In principle, Docker can run on any modern Linux system. However, a few specialized Linux distributions have emerged that propose to bring exactly what is needed to smoothly run Docker containers and nothing more. CoreOS is the most prominent one and was launched briefly after Docker. Redhat has reacted quickly and initiated project Atomic, which is developed in close cooperation with Redhat's own PaaS solution OpenShift. Canonical has only recently announced an own solution in this field called snappy Ubuntu core. It abandons traditional package managers and uses snappy, a new tool tailored for containerized apps. Boot2Docker is based on Tiny Core Linux and seems to address developers more than cloud hosters as it provides Windows and Mac OS X integration. OpenStack ships with CirrOS as a minimal image for virtual machines. However, CirrOS brings no Docker integration by default.

4.2 Image Registry

Docker uses layered images as a package format. Similar to a disk image of a virtual machine, the Docker image contains all the files necessary to run the container and do something meaningful. The image registry stores them and can be used to retrieve an image, if it is not already present on the host (r15). Docker Inc. provides a public image registry called Docker Hub (https://hub.docker.com) and an open source implementation for running a private registry. It is not a service registry (see service discovery). Dogestry is an alternative implementation using Amazon S3 compatible storage as a backend. The OpenStack counterpart of this category is Glance.

4.3 Container Management

Docker itself only provides a command-line interface (CLI) and a RESTful API for managing containers. This is fine for scripting and automating things, but there is still a need for a Web UI (r1, r2), e.g. for self-service administration by a customer. As the name implies, DockerUI provides exactly that missing WebUI for Docker, while the other candidates in this category provide additional functionality like management of composite applications (Panamax, r3) or broader management of containers and VMs (mist.io and Cockpit, r19). Direct terminal access to the containers via Web UI (r17) is currently under development by mist.io and already implemented by Rancher (see section 4.10). The OpenStack counterpart is Horizon.

4.4 Cluster Management

While Docker itself can only list and manage containers of a single host, a cluster management solution should allow the management of a cluster of Docker hosts and all containers on them, including the resource-aware placement of new containers (r6), automatic failover and migration of containers due to resource bottlenecks (Mills et al., 2011). We found four solutions providing parts of this functionality incl. a CLI (Apache Brooklyn, Citadel, CoreOS fleet and Docker Swarm). Decking is similar, but has additional orchestration capabilities (r3). Apache Mesos was originally dedicated to hosting solutions like Hadoop and Spark. Since version 0.20 it also supports running Docker containers. Other solutions like Shipyard build on them and provide a Web UI (r1, r2). Clocker additionally provides some orchestration (r3, r4) and networking functionality (r11), so that it is getting close to the management

Table 1: Overview of Docker software tools with fulfilled requirements (parentheses means partly fulfilled).

Software	Requirements
Image Registry	
Docker Hub	15
Dogestry	15
Container Mgmt	
Cockpit	1, 2, 19
DockerUI	1, 2
mist.io	1, 2, 7, 19, (17, 18)
Panamax	1, 2, 3
Cluster Management	
Brooklyn	(6)
Citadel	(6)
Clocker	3, 4, 11
Decking	3
Fleet	(6)
Flocker	3, 4, 6, 9, (11)
Mesos	(6)
Shipyard	1, 2
Swarm	(6)
Orchestration	
Compose	(3, 4)
Crane	
Fig	
Helios	3
Maestro	
Maestro NG	3
Shipper	3
Wire	3, 11
Service Discovery	
confd	10
Consul / Consul UI	8a
dnsmasq	8a

Software	Requirements
Service Discovery	
DoozerD	14
etcd	8a
Registrator	8b
SkyDNS	8a
SkyDock	8b
WeaveDNS	8a
Zookeeper	14
Software Defined Network	
Flannel	11, 12
Open vSwitch	11
Pipework	11, 12
Socketplane	11, 12
Weave	11, 12
Load Balancer	
HAProxy	13, 14
nginx	13, 14
Vulcan	13, 14
Monitoring	
cAdvisor	(18)
Grafana	(18)
Heapster	(18)
Kibana	18
logstash	18
Management Suites	
CF BOSH	3, 4, 6, (9, 18)
Flocker	3, 4, 6, 8, 9
Kubernetes	3, 4, 6, 8
Rancher	1, 2, (3, 4, 6, 9), 17, 18
OpenStack Docker Driver	1, 2, 3, 4, 6, 19, (9, 11)

suites (see section 4.10). Flocker doesn't provide a Web UI but also has additional functionality like basic orchestration and networking. It stands out due to its unique solution of linking storage to containers in a portable way (r9). Nova is kind of fulfilling this cluster management job in OpenStack, especially the Nova scheduler.

4.5 Orchestration

Service orchestration is an important feature for composite applications in an SaaS offering. When different components are deployed on different hosts to meet the scalability requirements, those separate deployments should appear as a single coherent subsystem to other components (Chauhan and Babar, 2011). BPEL and WSCI are examples of orchestration languages in SoA (Bucchiarone and Gnesi, 2006). Docker orchestration solutions mainly use YAML instead. Orchestration tools should be able to add links between Docker containers that are distributed across multiple hosts (r3). Some tools found in literature like Crane, Fig and Maestro (formerly Dockermix) are not able to do that and concentrate on single hosts. The developers of Helios, Maestro NG and Shipper all decided not to build upon cluster management solutions and instead connect to the different hosts on their own. All three come without a Web UI. Shipper seems to be the least mature of the three. Wire is an interesting tool, as it builds on Fig as well as Open vSwitch and dnsmasq to configure interdependent containers across hosts. The OpenStack counterpart of this category is Heat.

4.6 Service Discovery

Service discovery has always been an issue in SOA and has never been solved satisfactory in practice (Bachlechner et al., 2006). Recently, a new proposal was made for service discovery in a cloud context

based on OpenTosca, an Enterprise Service Bus and Chef (Vukojevic-Haupt et al., 2014). Within the Docker ecosystem, the proposed tools often represent more of a service registry and leave it up to the application developer to use the provided lookup mechanism (r10).

Etcd and Consul are two well-known representatives of this category. They provide a distributed key-value store in order to store ports, IP addresses or other characteristics of services running inside Docker containers. Zookeeper and DoozerD work in similar ways, but are less dedicated to Docker. Other tools like SkyDNS, dnsmasq and WeaveDNS try to solve the problem by reusing DNS for which there are discovery implementations in every OS. Tools like SkyDock or registrator automate the registration process (r8b) by monitoring Docker events and publishing information in the service registries. Confd stands out from the rest of the candidates, as it facilitates applications' usage of the configuration data from those service registries (r10). It reads data from service registries or environment variables and updates configuration files accordingly. In OpenStack, there is no dedicated service discovery tool, since it is focused on IaaS.

4.7 Software Defined Network

Within Docker, every container gets a private IP only visible on the same host. Ideally, an SDN is used to connect containers between multiple hosts (Costache et al., 2014, r11). Furthermore, isolation is beneficial, so that every customer (tenant) of the SaaS solution gets an own virtual network (Drutskoy et al., 2013). In an SDN, a logically centralized controller manages the collection of switches through a standard interface, letting the software control virtual and physical switches from different vendors (ibid.). Open vSwitch is a popular SDN solution that is also used by default in OpenStack's Neutron. It is also a central part of the larger OpenDaylight initiative. Socketplane and Pipework are overlay networks that make use of Open vSwitch and are tailored for Docker. They manage IP assignment (r12) and routing of messages between networks of multiple hosts. Flannel and Weave promise to do the same, but without support for Open vSwitch and therefore with less flexibility.

4.8 Load Balancer

The cloud can limit the scalability of a software load balancer due to security requirements. Amazon EC2 for example disabled many layer 2 capabilities, such as promiscuous mode and IP spoofing so that traditional techniques to scale software load balancers will not work (Liu and Wee, 2009). HAProxy and Nginx are forwarding traffic on layer 7 which limits scalability due to SSL termination (ibid.). However they are commonly used and fulfill our requirements (r13-14).

4.9 Monitoring

Monitoring is an essential part of cloud computing (Aceto et al., 2013). For monitoring Docker containers, you can use common solutions like Nagios that provide extensions for cloud scenarios, or some specialized tools like Sensu that are built for scalability from the ground up (Aceto et al., 2013). Within the Docker ecosystem, the most specialized solution is Google's cAdvisor, as it is tailored for monitoring containers. It brings its own Web UI. Logstash on the other hand is a general purpose tool for log file management (r18) and is often used in conjunction with elasticsearch as a NoSQL database and Kibana as a Web UI (Ward and Barker, 2014). Grafana is similar to Kibana and uses InfluxDB or other time series databases as data stores. It can be used in conjunction with cAdvisor since the latter can export data to InfluxDB. This seems advisable, since the Web UI of cAdvisor is limited to the latest data and does not show historical data. The combination can be further enhanced with Google Heapster which directly supports Kubernetes clusters. The container management solution mist.io does also include monitoring and seems to be the only one to support alerts based on thresholds (r7). Nagios is the default monitoring tool in OpenStack.

4.10 Management Suites

Suites are the most comprehensive tools in our review and at least include cluster management and orchestration capabilities (r3, r4, r6). They either build on multiple other solutions in order to cover the required functionality (e.g. Kubernetes, which relies on the CoreOS tools etcd, fleet and flannel) or are large monolithic solutions from the micro-service perspective (e.g. BOSH). Kubernetes is popular and further supports container migration (r8). The OpenStack Docker driver allows managing Docker containers just like KVM VMs in OpenStack and therefore reusing a large part of the OpenStack modules. In principle, this is the right way to go (r19). However, it does not match our use case very well (Docker inside KVM-based VMs), since you have to decide per host which hypervisor to run (KVM, Xen

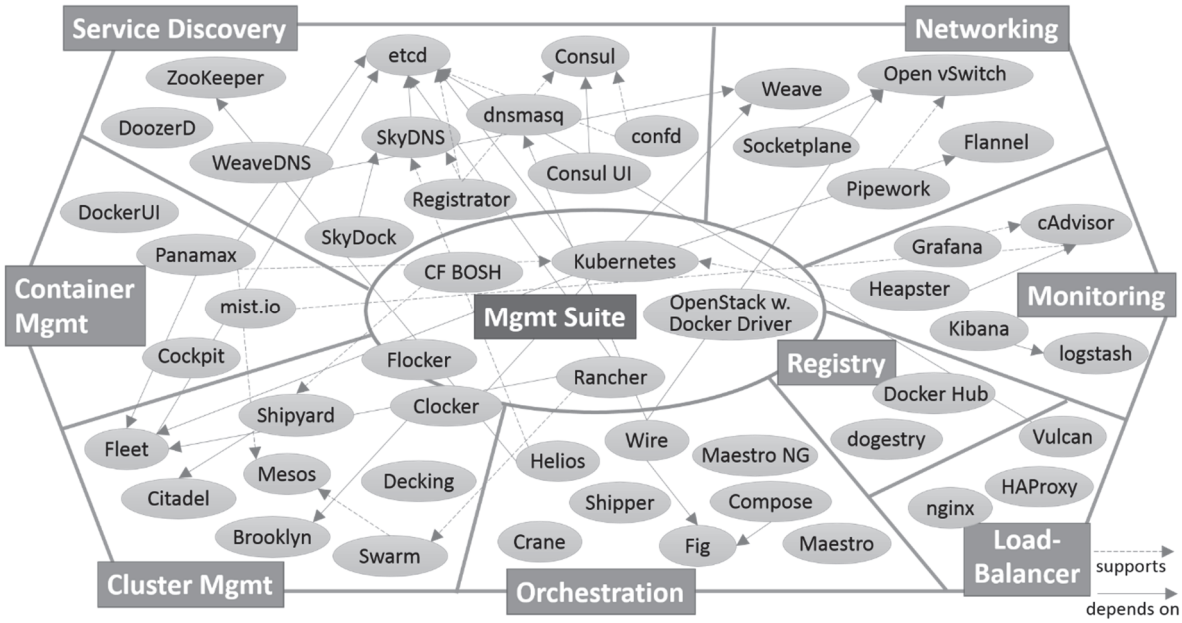

Figure 1: Docker ecosystem with dependencies (own illustration).

or Docker). Furthermore, not all modules are already updated to be used with Docker. A promising but still premature (alpha) candidate is Rancher.io. It aims at solving the multi-host problems of Docker by providing a Web-based UI, storage and networking capabilities. Version 0.3 from January 2015 allows starting and stopping containers on multiple hosts, linking containers across hosts and assigning storage (r9). They are also dedicated to support Docker Swarm and have a terminal agent that offers Web-based terminal sessions with containers (r17). BOSH is part of Pivotal's PaaS solution Cloud Foundry. It is able to start and stop containers on multiple hosts, but seems to be missing an overlay network component.

Figure 1 gives an overview of tools in the Docker ecosystem. The ellipses represent tools. The thick lines demarcate areas of functionality that are labeled in the rectangles. A position closer to the center of the figure within one category indicates that the tool has more functionality than others in the outer areas. Solid arrows represent dependencies between tools. Dashed arrows indicate that the tool is directly supported. It becomes obvious, that there already are some dependencies and interactions between tools. However, it is far from ideal and the most promising candidates of different categories are often developed side-by-side instead of hand-in-hand. Table 1 summarizes our findings more formally.

5 DISCUSSION

Despite the fact that the Docker ecosystem is huge, there still are requirements not fulfilled by any of the tools (r16, r20) and some are only fulfilled by a single tool (r7, r8b, r9). Many tools emerged quite recently and therefore must be considered premature.

Managing tenant data is maybe the most important missing part. (Lindner et al., 2010) argue, that there should be a complete supply chain for the cloud starting with deployment and monitoring and ending with accounting and billing. The economic part of this supply chain is currently not present in the Docker ecosystem. Updating a container can be emulated with a couple of Docker commands replacing it, since containers should be immutable. Still there should be a way to automate this. Registering a service in the registry is also a neglected requirement. Some tools do it, but our impression is that it is a better idea to use IP addresses and an SDN for routing instead of relying on one of the service discovery solutions when containers are migrated to another host. Storage is handled quite well by Flocker, but in our setup with Docker inside VMs there is still a problem. Volumes have to be mounted by the VM and mapped into the container. If the container moves, the volume has to be unmounted from the VM and mounted on the new host of the container. That means, that every container needs its own volume. If the space on the volume runs low,

growing it won't be easy. The Linux Device Mapper with its thin provisioning strategy can attenuate that problem but is not an ideal solution.

6 CONCLUSIONS

A Docker-based open source cloud environment to easily run composite applications as SaaS offerings would be a good basis for initiatives like the Open Cloud Alliance (Crisp Research, 2014) that aim at simplifying the process of bringing your applications to the cloud while preserving the freedom of choice and openness of the offering. In our paper, we have shown that many components are needed to fulfill the requirements for such a solution, which we dubbed runtime environment for SaaS applications (RaaS). It is similar to an IaaS environment, as we have shown with OpenStack, and includes some components from PaaS like load balancing and logging, but also has unique features like service orchestration and discovery. Not all requirements are currently fulfilled and despite first integration approaches, there is a need for closer cooperation within the Docker ecosystem. We plead for an embracing ecosystem project that serves as a coordination center for the tools that contribute to mastering the Docker management challenge. From our tests, Kubernetes with etcd, fleet and flannel seems the most usable combination right now. Mesos also seems a solid basis and integrations from other tools are currently in development (e.g. Compose/Swarm).

REFERENCES

Aceto, G., Botta, A., De Donato, W. and Pescapè, A. (2013), "Cloud monitoring: A survey", *Computer Networks*, Vol. 57 No. 9, pp. 2093–2115.

Bachlechner, D., Siorpaes, K., Fensel, D. and Toma, I. (2006), "Web service discovery-a reality check", *3rd European Semantic Web Conference*, Vol. 308.

Binz, T., Breitenbücher, U., Kopp, O. and Leymann, F. (2014), "TOSCA: Portable Automated Deployment and Management of Cloud Applications", *Advanced Web Services*, Springer, pp. 527–549.

Bucchiarone, A. and Gnesi, S. (2006), "A survey on services composition languages and models", *International Workshop on Web Services–Modeling and Testing (WS-MaTe 2006)*, p. 51.

Chauhan, M.A. and Babar, M.A. (2011), "Migrating service-oriented system to cloud computing: An experience report", *Cloud Computing (CLOUD) 2011, IEEE Int. Conf. on*, IEEE, pp. 404–411.

Cockcroft, A. (2014), "State of the Art in Microservices", DockerCon Europe 14, Amsterdam, The Netherlands.

Coffey, J., White, L., Wilde, N. and Simmons, S. (2010), "Locating software features in a SOA composite application", *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, IEEE, pp. 99–106.

Costache, C., Machidon, O., Mladin, A., Sandu, F. and Bocu, R. (2014), "Software-defined networking of Linux containers", *13th RoEduNet Conf.*, IEEE.

Crisp Research. (2014), *Open Cloud Alliance - Openness as an Imperative* (Strategy paper), Crisp Research, available at: http://bit.ly/1ArYcyc.

"Docker Ecosystem Mindmap". (n.d.). *MindMeister*, available at: http://bit.ly/1BjDgtW.

Docker, Inc. (2014), "About", *Docker Homepage*, available at: http://bit.ly/1OjEBLl.

Drutskoy, D., Keller, E. and Rexford, J. (2013), "Scalable network virtualization in software-defined networks", *Internet Computing, IEEE*, Vol. 17 No. 2, pp. 20–27.

Evans, E. (2003), *Domain driven design: Tackling Complexity in the Heart of Software*, Addison-Wesley, Boston.

Jain, R. and Paul, S. (2013), "Network virtualization and software defined networking for cloud computing: a survey", *Communications Magazine, IEEE*, Vol. 51 No. 11, pp. 24–31.

Koukis, V. (2013), "Flexible storage for HPC clouds with Archipelago and Ceph", *8th Workshop on Virtualization in High-Performance Cloud Computing*, ACM.

Kratzke, N. (2014), "Lightweight Virtualization Cluster How to Overcome Cloud Vendor Lock-In", *J. of Computer and Communications*, Vol. 2 No. 12, pp. 1–7.

Lindner, M., Galán, F., Chapman, C., Clayman, S., Henriksson, D. and Elmroth, E. (2010), "The cloud supply chain: A framework for information, monitoring, accounting and billing", *2nd Int. Conf. on Cloud Computing*.

Liu, H. and Wee, S. (2009), "Web server farm in the cloud: Performance evaluation and dynamic architecture", *Cloud Computing*, Springer, pp. 369–380.

Mietzner, R., Leymann, F. and Papazoglou, M.P. (2008), "Defining composite configurable SaaS application packages using SCA, variability descriptors and multi-tenancy patterns", *ICIW 2008*, IEEE.

Mills, K., Filliben, J. and Dabrowski, C. (2011), "Comparing VM-placement algorithms for on-demand clouds", *Cloud Computing Technology and Science (CloudCom), IEEE 3rd Int. Conf. on*, IEEE, pp. 91–98.

Papazoglou, M.P. (2003), "Service-oriented computing: Concepts, characteristics and directions", *Web Information Systems Engineering (WISE 2003). 4th Int. Conf. on*, IEEE, pp. 3–12.

Peinl, R. (2015), "Docker ecosystem on Google Docs", available at: http://bit.ly/1DJ0eS4.

Rosen, R. (2014), "Linux containers and the future cloud", *Linux Journal*, Vol. 2014 No. 240, p. 3.

Roßbach, P. (2014), "Docker Poster", *Entwickler Magazin Docker spezial*, Vol. 2014 No. Docker spezial.

Scheepers, M.J. (2014), "Virtualization and Containerization of Application Infrastructure: A Comparison", 21st Twente Student Conference on IT,

University of Twente, Twente, The Netherlands.
Sefraoui, O., Aissaoui, M. and Eleuldj, M. (2012), "OpenStack: toward an open-source solution for cloud computing", *International Journal of Computer Applications*, Vol. 55 No. 3, pp. 38–42.
Turnbull, J. (2014), *The Docker Book: Containerization is the new virtualization*, James Turnbull.
Vukojevic-Haupt, K., Haupt, F., Karastoyanova, D. and Leymann, F. (2014), "Service Selection for On-demand Provisioned Services", *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*, IEEE, pp. 120–127.
Ward, J.S. and Barker, A. (2014), "Observing the clouds: a survey and taxonomy of cloud monitoring", *Journal of Cloud Computing: Advances, Systems and Applications*, Vol. 3 No. 1, p. 40.