J Grid Computing (2016) 14:265–282
DOI 10.1007/s10723-016-9366-y

# Docker Cluster Management for the Cloud - Survey Results and Own Solution

**René Peinl · Florian Holzschuher · Florian Pfitzer**

Received: 27 October 2015 / Accepted: 31 March 2016 / Published online: 13 April 2016


**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 close some of these gaps with our own integration components and tool enhancements, resulting in the currently most complete management suite.

**Keywords** Cloud computing · Management tools · Microservices · System integration · Docker · Container

## 1 Introduction

Virtual machines as the core virtualization construct of the cloud have been improved successively by addressing scheduling, packaging and resource access problems [1]. VM instances use isolated large files on their host to store their entire file system and typically run as a single process on the host. A full guest OS image for each VM is needed in addition to the binaries and libraries necessary for the applications. That leads to high RAM and disk storage requirements and slow startup (ibid.).

Containers are a more lightweight virtualization concept, i.e. less resource and time consuming [2, 3]. They can be seen as more flexible tools for packaging, delivering and orchestrating both software services and applications. Containers build on recent advances in virtualization and therefore allow for better portability and interoperability while still utilizing operating systems virtualization techniques [4].

Although lightweight operating system (OS) virtualization techniques like Solaris Zones and OpenVZ are long established, it was the release of Docker in March 2013 [5] that led to mass adoption and even a hype around containerization [6]. It originally built on LXC but then Docker, Inc. wrote an own runtime called libcontainer in order to easen Go language bindings. When Docker, Inc. started to build their own ecosystem around Docker, CoreOS placed rkt (formerly rocket) against Docker arguing that Docker stopped concentrating on building the best possible container runtime and did not engage in the security problems as much as needed [7]. With LXD from Ubuntu there is a third alternative for running containers, although Ubuntu claims to concentrate providing

R. Peinl (✉) · F. Holzschuher · F. Pfitzer
Institute of Information Systems, Hof University,
Alfons-Goppel-Platz 1, Hof, Germany
e-mail: rene.peinl@hof-university.de

F. Holzschuher
e-mail: florian.holzschuher2@iisys.de

F. Pfitzer
e-mail: florian@pfitzer.me

a lightweight approach for virtualizing a whole operating system whereas Docker concentrates on packaging applications (ibid.).

In the meanwhile, the so called Open Container Initiative tries to bring all players together again and is working on a container format specification and a runtime implementation called runC [8].

Docker aims at making container technologies easy to use and among other things encourages a service-oriented architecture (SOA) and especially the microservice architecture style [9, 10]. 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 [11] was already discussed in [12]. 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 [5], including links to dependent containers on the same host. Microservices can be briefly summarized as a "loosely coupled service oriented architecture with bounded contexts" [13], with loosely coupled denoting 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 [14, cf.]. Enterprise applications are often complex composite applications, which consist of multiple individual components [15] and therefore match microservices well. However, the Docker tools soon reach their limits when it comes to managing containers in a cluster or creating links across multiple hosts [6]. To overcome those, a myriad of tools is currently in development. We found over 60 tools in "The Docker Book" [9], a special issue of the German "developer magazine" dedicated to Docker [16] and the "Docker ecosystem" mindmap [17] with relevance for building an automated Docker cluster management solution similar to OpenStack on the IaaS level [18, for a full list]. Docker Inc even counts 50,000 third-party projects on GitHub that are using Docker [19]. Recent publications like [20–22] show the ongoing interest about this topic. 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.

## 2 Methodology

We followed a similar procedure to [23]. The goal in our project is to move existing open source systems to the cloud in order to offer them as SaaS. They form a complex composite application with lots of dependencies. Therefore, an infrastructure solution is needed that allows administrators specifying the dependencies and takes care of running the systems. An analysis of the use case and deriving the relevant requirements were the first steps. Then, a list of tools was collected and analysed based on available documentation. The most promising candidates were installed and tested in more detail. The interplay between different tools that cover parts of the requirements was given special attention. Following this approach, two solution stacks were identified. One of the stacks was enhanced with own glue code, so that the integration between the tools became better. This solution is finally evaluated against the requirements.

Although this procedure was project specific, the results can be transferred to all companies running Docker in a cloud for themselves or other companies and having at least a medium number of containers to run (several dozen). Docker itself targets different user groups from single developers who want to easily test their applications to large cloud providers like Google launching millions of containers per month. The solution described here targets a large spectrum of users in between those two extremes. As soon as you talk of true cloud environments with elasticity and on-demand provisioning, you need automation and then all the requirements discussed below need to be fulfilled.

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 present results of an empirical study that tested the most promising candidates for a comprehensive solution and propose our own solution based on a number of existing tools. We conclude with remaining challenges and an outlook.

## 3 Case Description

The goal of the SCHub project (Social Collaboration Hub, funded by the BMBF as part of the FHprof-Unt 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 microservice approach [13]. It therefore represents a composite application [24]. The solution will be available as Software as a Service (SaaS) in the cloud as well as for on premise installation. In order to achieve 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, dedicated virtual machines (VM) per customer are used additionally. These VMs become the Docker hosts in this case. OpenStack serves as the basis [25]. 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 [26] 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 [27], database as a service and so on, so that typical Container as a Service offerings [28] do not provide enough. 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.

## 4 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 [15]. Many of the required features are simply a transfer of IaaS management features to Docker. There should be a central list of containers (r01) with an overview of resource usage, IP address, open ports, dependencies and so on. You need a detail view of a container (r02) 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 (r04) that includes all the dependencies and to instantiate the whole solution instead of single containers (r05).

For doing so, the management solution should monitor resource usage of hosts (r07) and automatically choose one with free resources, based on an editable placement strategy (r06). 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 alerts (r08) which in turn can trigger actions like migrating a container to another host. Migration (r10) could be performed by stopping the container, unmounting the storage, starting an identical container on a new host, updating service references (r11) and mounting the storage (r12) there. Besides storage, there should also be an easy way to pass configuration data to the application inside the container (r13). This data has to be stored in a distributed key/value store (r14).

For communication between containers across hosts (r15), you ideally need an overlay network or a software defined network (SDN) [29]. Its configuration should be accessible directly from the Docker management UI, e.g., for defining IP address ranges

(r16). The Web UI of the SDN could be simply integrated. You need routing of external requests with URLs to tenant-specific container IPs (r17). This routing should include load balancing if multiple container instances are available (r18). There should be a way to review the list of available images (r19) 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 (r20), 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 (r21) and review the log files (r09), both using the Web UI (Table 1).

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 (r03), since container solutions are often built on IaaS platforms [28]. Finally, there should be an integration with a tenant / customer management solution (r22) 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 authenticating an administrator anyway.

**Table 1** List of requirements for a container mgmt. solution

| No. | Requirement description |
|---|---|
| r01 | Container list |
| r02 | Container details view/edit |
| r03 | Create new Docker hosts (VMs) from Web |
| r04 | Blueprint for composite applications |
| r05 | Launch (composite) application |
| r06 | Editable container placement strategy |
| r07 | Monitor container & host resource usage |
| r08 | Trigger alerts based on monitoring events |
| r09 | Review log files from Web UI |
| r10 | Migration of containers to different hosts |
| r11 | Update service references |
| r12 | Mount (external) storage volumes |
| r13 | Pass configuration data to containers |
| r14 | Store data in a distributed database |
| r15 | Communication of containers across hosts |
| r16 | Web-based configuration for network |
| r17 | Routing of external URLs to internal IPs |
| r18 | Load balancing between containers |
| r19 | List available images and instances |
| r20 | Update containers with new image |
| r21 | Access container command line from Web |
| r22 | Customer/tenant management solution |

## 5 Existing Solutions

Weve 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. Figure 1 gives an overview of tools in the Docker ecosystem, which are described in more detail in the next sections. 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. The subsections start with a general introduction about the respective category of tools and what to expect from a representative in this area and then continues with the description of concrete tools. At the end of each subsection we point at similarities to OpenStack, where applicable.

### 5.1 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 its own solution in this field called snappy

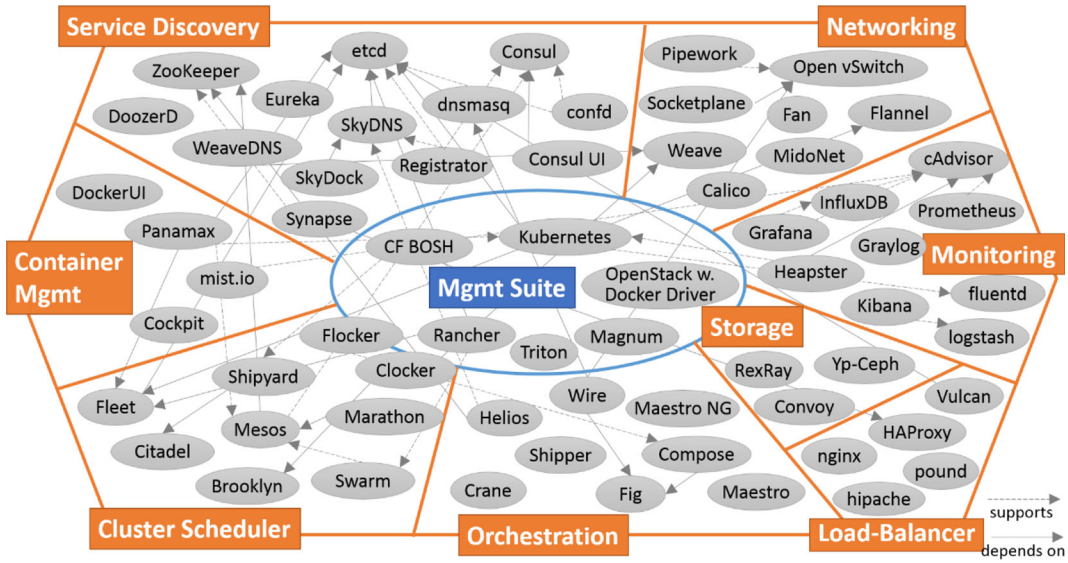


**Fig. 1** Docker ecosystem with dependencies (own illustration)

Ubuntu core. It abandons traditional package managers and uses snappy, a new tool tailored for containerized apps. Even VMware (Photon) and Intel (Clear Linux), both not really well-known for open source solutions, offer own optimized container host Linux distributions. 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.

### 5.2 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 (r19).

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). Recently, version 2.0 of Docker Hub and the image registry were released (http://bit.ly/1GOk5N0) with a new architecture and additional features like verifiable images, resumable downloads and a pluggable storage backend with S3 support [30]. Dogestry is an alternative implementation using Amazon S3 compatible storage as a backend. Google also offers a hosted container registry for its own cloud platform, as do Amazon (EC container registry) and CoreOS (quay.io) [31].

The OpenStack counterpart of this category is Glance.

### 5.3 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 still is a need for a Web UI (r01, r02), 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, r04) or broader management of containers and VMs (mist.io and Cockpit, r03). Direct terminal access to the containers via Web UI (r21) is currently under development by mist.io and already implemented by Rancher (see Section 5.11).

The OpenStack counterpart is Horizon.

### 5.4 Cluster Scheduler

While Docker itself can only list and manage containers of a single host, a cluster scheduler should allow the management of a cluster of Docker hosts and all containers on them, including the resource-aware placement of new containers (r06) and automatic failover and migration of containers due to resource bottlenecks [32]. The need for such a solution first arose in large companies running hundreds and thousands of containers like Google [33]. Some of the solutions in this category are inspired by Google Borg or Omega [34].

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 (r04). 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 (r01, r02). Clocker additionally provides some orchestration (r04, r05) and networking functionality (r15), so that it is getting close to the management suites (see Section 5.11). 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 (r12).

Nova is kind of fulfilling this cluster management job in OpenStack, especially the Nova scheduler.

### 5.5 Orchestration

Service orchestration is an important feature for composite applications in a 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 [23]. BPEL and WSCI are examples of orchestration languages in SoA [35]. Caballer et al. propose a Resource and Application Description Language (RADL), for dynamically deploying virtual machine images to different cloud provider [36]. Docker orchestration solutions mainly use YAML instead. Orchestration tools should be able to add links between Docker containers that are distributed across multiple hosts (r04).

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.

### 5.6 Service Discovery

Service discovery has always been an issue in SOA and has never been solved satisfactory in practice [37]. Recently, a new proposal was made for service discovery in a cloud context based on OpenTosca, an Enterprise Service Bus and Chef [38]. 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 (r14). Docker's own mechanism of container linking (only on the same host) uses environment variables to pass the IP addresses of linked containers. A third alternative is to use reverse proxies to transparently route IP requests to the respective services. An advantage of that solution is that you can achieve load balancing at the same time. The downside is that routing on a lower layer (e.g., 3) of the network stack is considered faster than on layer 7. Proxy solutions are discussed in Section 5.9.

Etcd and Consul are two well-known representatives of the service registry 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. Eureka from Netflix and Synapse from AirBnB are two examples of tools that were open sourced by large cloud service providers. Eureka is specially designed to run in the AWS cloud. Synapse leverages HAproxy to route requests to services and watchers to get updated about changes in services' addresses [39]. Tools like SkyDock or registrator automate the

registration process (r11) 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 (r13). 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.

### 5.7 Storage Volumes

Storage is a critical topic in Docker applications. Due to the layered file system (AUFS or OverlayFS), write performance inside Docker containers is far worse than usually. Therefore, Docker suggests using volumes mounted from the host for writing or attaching specialized storage containers that are not using a layered file system. However, both solutions don't cover the requirement (r12) of mounting external storage like Amazon EBS (elastic block storage) or Ceph directly to the container instead of mounting them to the host and afterwards to the container.

Flocker was the only solution for that kind of problem for quite some time. It is leveraging the capabilities of ZFS and synchronizes data between hosts [40]. Starting with version 1.7 in July 2015, Docker provides a new plug-in mechanism for both storage volumes and networking. So called volume drivers allow Docker to mount different kinds of storage volumes. ClusterHQ, the company behind Flocker, partnered with EMC [41] to bring the EMC storage solutions ScaleIO and XtremIO to Docker as well as Amazon EBS, Rackspace Cloud Block Storage and OpenStack Cinder compatible storage providers like Ceph. Other solutions quickly followed this way, so that four months after the public announcement, there are already some competing solutions. EMC provides its own open source solution called REX-Ray. The Ceph RBD Docker volume driver by Yp engineering is one of three Ceph-based solutions [42]. Rancher Convoy can use NFS or the Linux Device Mapper as a storage backend. It runs on individual hosts and requires a central object store in order to backup a volume to that object store and restore it on a different host. This requires two copy operations over the network and is therefore not suitable for migrating container from one host to another. However, even the Ceph-based solutions have open issues like how to determine the "right" size of the assigned block storage [43], since it usually cannot be resized online, so that auto-scaling means a few seconds of downtime.

The OpenStack counterpart of this category is Cinder.

### 5.8 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 [43, r15]. Furthermore, isolation is beneficial, so that every customer (tenant) of the SaaS solution is getting her own virtual network [45]. 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. Despite that, there is a growing number of experts that point to the limitations of Open vSwitch's scalability, since the number of tunnels it establishes is growing with the square of the number of hosts. Socketplane and Pipework are overlay networks that make use of Open vSwitch and are tailored for Docker. They manage IP assignment (r16) and routing of messages between networks of multiple hosts. Socketplane itself runs in a Docker container and uses Consul to make the configuration available to every host. The downside is that the Consul server has to run internally as part of Socketplane which is using an outdated version. It is not possible to use an existing Consul cluster.

Flannel and Weave promise to do the same, but without support for Open vSwitch. Flannel can leverage VXLAN (virtual extensible local area network) to tunnel packets or rewrite transport routes on the host or in an Amazon virtual private cloud (VPC). It uses an etcd cluster to share configuration data and IP addresses. Weave is the only SDN solution that can encrypt the network, which may be relevant in some setups. It runs in a Docker container, which leads to a strange dependency situation: if Docker should use the Weave bridge, Weave has to be started first to create it and Weave needs the Docker daemon to start its

container. Therefore, the bridge has to be created manually. Weave can handle the IP address assignment on its own and provide a single subnet spanning all hosts or let the Docker daemon do this and assign individual subnets to all hosts. Weave encapsulates network packets in its own UDP-based protocol.

Socketplane was bought by Docker, Inc. in March 2015 and was not updated since then. Instead, Docker introduced a pluggable network layer in Docker 1.7 and is working on libnetwork, a host-to-host communication solution that uses network drivers to accomplish the goal. This attracted new solutions like project Calico which uses layer 3 routing together with BGP (border gateway protocol) and therefore does not require an overlay network and, using suitable hardware, no tunneling or network address translation. It can be used both for OpenStack and for Docker.

MidoNet by Midokura is somehow similar, but even more OpenStack focused. Additionally, it has a commercial branch and uses Zoekeeper as well as Cassandra. Similar to Calico, it uses the BGP to route between hosts.

Another recent alternative is Ubuntu Fan. Instead of sharing routes and IP addresses in the cluster, it uses a deterministic algorithm to create container IP addresses out of the host IP address. IP packets are encapsulated into other IP packets so that they can be routed to the target host, which can remove the surrounding packet and deliver the original packet to the right container. The drawback is that containers will always get a host-specific IP address, which cannot be moved together with a container.

The OpenStack counterpart is Neutron.

### 5.9 Load Balancer

"Load Balancing is an integral part of Cloud Computing and elastic scalability" [46]. 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 [47].

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 (r17-r18). More specialized tools for this purpose include hipache and pound, that both work as reverse proxy and http load balancer. Hipache is also able to deal with Web socket connections and can be easily scaled horizontally.

Load Balancing as a Service is an advanced service of the OpenStack Neutron component.

### 5.10 Monitoring

Monitoring is an essential part of cloud computing [48]. It should provide both historical and timely information about resource usage from hardware to virtual machine and container level. Monitoring application level key performance indicators regarding conformance to service level agreements and as a basis for pay-per-use would be desirable as well (ibid.), but is neglected here. Typically, solutions are split into tools that collect and store data about resource consumption and Web-based front-ends for visualizing them. 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 [48].

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. FluentD, Graylog 2 and Logstash are general purpose tools for log file management (r09). The latter is often used in conjunction with Elasticsearch as a NoSQL database and Kibana as a Web UI (ELK Stack) [49]. 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 and recently Prometheus. This seems advisable, since the Web UI of cAdvisor is limited to the latest data of a single host 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 (r08).

Nagios is the default monitoring tool in OpenStack.

### 5.11 Management Suites

Suites are the most comprehensive tools in our review and should at least include cluster scheduler and orchestration capabilities (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 microservice perspective (e.g. BOSH). They aim at what Kratzke calls a lightweight virtualization cluster [6].

Kubernetes is a popular choice in this category being driven by Google and CoreOS. It is the suite which fulfills the most requirements of all tools (see Table 2). 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 (r03). However, it does not match our use case

**Table 2** Overview of Docker software tools with fulfilled requirements (light grey means partly fulfilled)

| Category | Software | r01 | r02 | r03 | r04 | r05 | r06 | r07 | r08 | r09 | r10 | r11 | r12 | r13 | r14 | r15 | r16 | r17 | r18 | r19 | r20 | r21 | r22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Image Registry | Docker Hub | | | | | | | | | | | | | | | | | | | ■ | | | |
| | Dogestry | | | | | | | | | | | | | | | | | | | ■ | | | |
| | Google CR | | | | | | | | | | | | | | | | | | | ■ | | | |
| | EC2 CR | | | | | | | | | | | | | | | | | | | ■ | | | |
| | quay.io | | | | | | | | | | | | | | | | | | | ■ | | | |
| Container Mgmt | Cockpit | ■ | ■ | ■ | | | | | | | | | | | | | | | | | | | |
| | DockerUI | ■ | ■ | | | | | | | | | | | | | | | | | | | | |
| | mist.io | ■ | ■ | ■ | | | | | ■ | ▒ | | | | | | | | | | | | ▒ | |
| | Panamax | ■ | ■ | | ■ | | | | | | | | | | | | | | | | | | |
| Cluster Scheduler | Brooklyn | | | | | | ▒ | | | | | | | | | | | | | | | | |
| | Citadel | | | | | | ▒ | | | | | | | | | | | | | | | | |
| | Clocker | | | | ■ | ■ | | | | | | | | | | ■ | | | | | | | |
| | Decking | | | | ■ | | | | | | | | | | | | | | | | | | |
| | Fleet | | | | | | ▒ | | | | | | | | | | | | | | | | |
| | Mesos + Marathon | | | | ■ | ■ | ■ | | | | | | | | | | | | | | | | |
| | Shipyard | ■ | ■ | | | | | | | | | | | | | | | | | | | | |
| | Swarm | | | | | | ▒ | | | | | | | | | | | | | | | | |
| Orchestration | Compose | | | | ▒ | ▒ | | | | | | | ■ | | | | | | | | | | |
| | Crane | | | | | | | | | | | | | | | | | | | | | | |
| | Fig | | | | | | | | | | | | | | | | | | | | | | |
| | Helios | | | | ■ | | | | | | | | | | | | | | | | | | |
| | Maestro | | | | | | | | | | | | | | | | | | | | | | |
| | Maestro NG | | | | ■ | | | | | | | | | | | | | | | | | | |
| | Shipper | | | | ■ | | | | | | | | | | | | | | | | | | |
| | Wire | | | | ■ | | | | | | | | | | ■ | ■ | | | | | | | |
| Service Discovery | confd | | | | | | | | | | | | | ■ | | | | | | | | | |
| | Consul / Consul UI | | | | | | | | | | | | | | ■ | | | | | | | | |
| | dnsmasq | | | | | | | | | | | | | | ■ | | | | | | | | |
| | DoozerD | | | | | | | | | | | | | | ■ | | | | | | | | |
| | etcd | | | | | | | | | | | | | | ■ | | | | | | | | |
| | Eureka | | | | | | | | | | | | | | ■ | | | | | | | | |
| | Registrator | | | | | | | | | | | ■ | | | | | | | | | | | |
| | SkyDNS | | | | | | | | | | | | | | ■ | | | | | | | | |
| | SkyDock | | | | | | | | | | | ■ | | | | | | | | | | | |
| | Synapse | | | | | | | | | | | | | | ■ | | | | | | | | |
| | WeaveDNS | | | | | | | | | | | | | | ■ | | | | | | | | |
| | Zookeeper | | | | | | | | | | | | | | ■ | | | | | | | | |

**Table 2** (continued)

| Category | Software | r01 | r02 | r03 | r04 | r05 | r06 | r07 | r08 | r09 | r10 | r11 | r12 | r13 | r14 | r15 | r16 | r17 | r18 | r19 | r20 | r21 | r22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Storage | Convoy | | | | | | | | | | | | | | | | | | | | | | |
| | REX-Ray | | | | | | | | | | | | | | | | | | | | | | |
| | Yp Ceph | | | | | | | | | | | | | | | | | | | | | | |
| SDN | Flannel | | | | | | | | | | | | | | | | | | | | | | |
| | MidoNet | | | | | | | | | | | | | | | | | | | | | | |
| | Open vSwitch | | | | | | | | | | | | | | | | | | | | | | |
| | Pipework | | | | | | | | | | | | | | | | | | | | | | |
| | Project Calico | | | | | | | | | | | | | | | | | | | | | | |
| | Socketplane | | | | | | | | | | | | | | | | | | | | | | |
| | Ubuntu Fan | | | | | | | | | | | | | | | | | | | | | | |
| | Weave | | | | | | | | | | | | | | | | | | | | | | |
| Load Balancer | HAProxy | | | | | | | | | | | | | | | | | | | | | | |
| | hipache | | | | | | | | | | | | | | | | | | | | | | |
| | Nginx | | | | | | | | | | | | | | | | | | | | | | |
| | pound | | | | | | | | | | | | | | | | | | | | | | |
| | Vulcan | | | | | | | | | | | | | | | | | | | | | | |
| Monitoring | cAdvisor | | | | | | | | | | | | | | | | | | | | | | |
| | Grafana | | | | | | | | | | | | | | | | | | | | | | |
| | Heapster | | | | | | | | | | | | | | | | | | | | | | |
| | Kibana | | | | | | | | | | | | | | | | | | | | | | |
| | Fluentd | | | | | | | | | | | | | | | | | | | | | | |
| | Graylog2 | | | | | | | | | | | | | | | | | | | | | | |
| | logstash | | | | | | | | | | | | | | | | | | | | | | |
| | Prometheus | | | | | | | - | - | | | | | | | | | | | | | | |
| | InfluxDB | | | | | | | - | - | | | | | | | | | | | | | | |
| Management Suites | CF BOSH | | | | | | | | | | | | | | | | | | | | | | |
| | Flocker | | | | | | | | | | | | | | | | | | | | | | |
| | Kubernetes | | | | | | | | | | | | | | | | | | | | | | |
| | OpenStack w. Docker Driver | | | | | | | | | | | | | | | | | | | | | | |
| | Magnum | | | | | | | | | | | | | | | | | | | | | | |
| | Rancher | | | | | | | | | | | | | | | | | | | | | | |
| | Triton | | | | | | | | | | | | | | | | | | | | | | |

very well (Docker inside KVM-based VMs), since you have to decide per host which hypervisor to run (KVM, Xen 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 (r12). They are also dedicated to support Docker Swarm and have a terminal agent that offers Web-based terminal sessions with containers (r21). The development of Rancher is quite active with version 0.42 being published on 20$^{th}$ of October 2015. Interesting new features include the integration of Docker machine in order to create new Docker hosts from the same UI (r03). 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.

Joyent's Triton is going a completely different way and brings Docker containers to the SmartOS operating system which is an enhanced version of Sun's Solaris. They managed to run Docker containers with Linux guests inside this Unix system with strong isolation making VMs around containers superfluous. On the downside, they had to rewrite the Docker engine on SmartOS and therefore are not 100% compatible (http://bit.ly/1RzO6pr) and maybe won't ever be given the development speed of Docker. Table 2 summarizes our findings in a formal way.

### 5.12 Customer Management

A missing piece that is seldom discussed as part of the Docker ecosystem is the customer side of the story, although it is a common requirement that a cloud service provider needs both a way to register customers, upload users (e.g., via LDIF or SCIM) and create bills. A customer administration center should therefore provide a rating, charging and billing (RCB) solution as well as a limited access to monitoring in order to observe compliance with SLAs. JBilling is an example of a billing solution for cloud services. CloudKitty is an RCB solution specifically tailored for OpenStack and integrates with Ceilometer and Horizon, whereas Cyclops is suitable for different cloud platforms as well as SaaS models.

## 6 In Depth Analysis

The choice of candidates for the in-depth analysis is based on the criteria "number of supported features" and "active development and community". Therefore, Clocker and CF BOSH were not analyzed in detail, since they haven't shown an active development in 2015. Rancher provides a convenient Web interface, but lacks cluster functionality beyond host-spanning networking. It is also under heavy development with a warning from the developers about upgrades not being supported and possible breaking changes before reaching version 1.0. The Docker driver for OpenStack would have been a favorite solution, but it does not support the use case with containers inside VMs. It allows only Docker instead of VMs and the administrator has to choose per host, whether Docker or KVM is used. The OpenStack Magnum project seems promising, but is very premature at the moment. We did not succeed in getting it fully operational.

Both Docker, Inc. and CoreOS provide a number of integrated tools that allow basic management of a cluster solution. Docker, Inc. has *Docker machine* to prepare hosts to serve as Docker host. *Docker compose* can be used to describe composite applications consisting of multiple containers, resources and links between them. With *Docker swarm,* these composite applications can be started on multiple hosts. *Libnetwork*, the default implementation for the network plugins and the successor of Socketpipe allows for cluster-wide communication between containers. However, the suite lacks a central Web-UI, monitoring and service discovery features. Some of that is provided by Docker Datacenter, which literally was announced in the last possible minute to be included in this paper [50].

CoreOS has fleet as a cluster scheduler, flannel as an overlay network and etcd as a distributed service registry. They even provide an alternative to the Docker runtime called rkt, which was recently published in version 1.0 and claims to be more secure than Docker [51]. However, the real cluster solution is missing something like Kubernetes, which CoreOS acknowledges by offering Tectonic, a commercial grade package of Kubernetes, the CoreOS tools and enterprise support.

That leaves two solutions that are both relatively mature and already well integrated with a number of other Docker tools: Kubernetes and Mesos. While Kubernetes comes pretty much pre-integrated with the CoreOS tools flannel, etcd and fleet, Mesos does not prescribe components for service discovery, multi-host networking and orchestration. Our proposal for the Mesos stack is based on functional evaluation and the idea of building on well-established components. It cannot be denied that forming a counterpart to the Kubernetes stack was also leading the selection. This was due to the fear that further developments of the CoreOS tools like etcd would concentrate on playing well together with Kubernetes and neglecting the integration to other tools like Mesos.

### 6.1 Kubernetes

Kubernetes not only uses its own terminology, but also additional concepts. A Kubernetes cluster consists of at least one master. Multi-master operation and therefore fail-safety was introduced in version 1.0 in July 2015 (current version is 1.1.7 from January 2016). Its service called podmaster is responsible for leader election (http://bit.ly/1LLyQWy). The Docker hosts inside a Kubernetes cluster are called nodes (formerly minions). Each node can run pods, which represent an application and consist of one or multiple Docker containers. A pod is the smallest manageable unit. Pods are automatically assigned to nodes. This assignment can be influenced with labels, so that applications with high IOPS (input/output operations per second) can be assigned to a node with SSDs for example. Pods are started using Kubelets. A Kubelet is an agent running on every node and can restart a pod if it fails. If a whole node fails, Kubernetes needs an additional replication controller to restart the pods from the failed node on a different node. Kubernetes accesses pods via services, which can be seen as proxies with their own virtual IP address. They can also be used as a load balancer, to e.g. use three pods together as a single load-balanced application. A service can be discovered using DNS or environment variables. Pods communicate with each other using flannel. Each node gets its own subnet (/24 by default) so that it can run 254 pods at most. The IP address of a pod never changes. The pods are defined using JSON (JavaScript object notation) or YAML files (yet another markup language). Regarding storage, Kubernetes allows to use block devices from Google Cloud Engine instead of local disks from the minion. Since version 1.0 it additionally supports NFS (network file system), iSCSI, GlusterFS, Ceph, and Amazon EBS (elastic block storage).

### 6.2 Mesos

Apache Mesos is not focused on Docker, but is a cluster management platform that abstracts resources from single hosts and provides them as a flexible resource pool to so called frameworks running on top of Mesos. It has its roots in the big data area and plays well together with Hadoop and Spark, but also with Cassandra, Elasticsearch and Docker. For Docker management, both Aurora and Marathon can be used as frameworks for scheduling Docker-based workloads. However, Aurora introduces a dependency to Python, which renders most of the official Docker images useless. Therefore, Marathon was the obvious choice here. Mesos uses Zoekeeper for leader election among the master nodes. Compute nodes are called slaves in Mesos. The framework (Marathon in our case) is responsible for placing Docker containers on slaves. Mesos is offering free resources from the slaves to the framework. Communication between the Mesos master and the framework is using Google's ProtoBuf protocol. Mesos uses a bin-packing strategy to utilize slaves which already have workloads as completely as possible before placing workloads on a completely free slave. The placement can be influenced with constraints that allow more sophisticated strategies than Kubernetes' labels. The constraints unique (e.g., only once per slave or data center), group_by (e.g., equally distributed over all racks), cluster (e.g. only on slaves with SSDs) as well as like and unlike are available. The latter two evaluate regular expressions for placement.

Mesos does also allow the use of different frameworks that jointly work on a single resource pool. The workloads are started by an executor service running on each slave. Marathon uses JSON files to describe applications. It contains all Docker-specific parameters except the name of the container, which is auto-generated by Mesos. Composite applications can be defined as groups of applications. These groups can contain dependencies, which Marathon considers when starting the containers.

Service discovery is not solved in Marathon itself. Two solutions are recommended, which both have drawbacks. Marathon can update a HAproxy instance once a minute or use Mesos DNS for service discovery. Both solutions lead to a considerable delay in service availability.

Marathon does also support a basic form of rolling updates. If a container image is updated, a configurable part of old container instances is stopped (0-100%) before new instances with the updated container are launched. Then a health check is performed. If the new instances are running well, the remaining instances are stopped and started step by step. A drawback of the current Marathon version is, that errors during the deployment, like a missing container

image or a container that terminates with errors during the start do not lead to proper errors in Marathon and Mesos. Therefore, there is also no automatic error handling or notification. The administrator has to manually recover the system from the failure.

### 6.3 Discussion

Although Kubernetes has a few more pre-configured pieces directly built-in, none of the solutions fulfill all the requirements. We chose Mesos as a basis for own integration developments since it seems more open for cooperating with different tools than Kubernetes and also less commercially oriented than CoreOS with Tectonic, although it is sad to notice that the extended version of Mesos, the Data Center Operating System, is only available for free on Amazon AWS but not for on premise installations.

The additional components needed were chosen based on interoperability with Docker's latest plug-in model, matureness and feature richness. Figure 2 provides an overview including the control flow.

For service discovery, the choice was between etcd and Consul. Zookeeper is already running for master election in the Mesos cluster, but only on the master nodes and we didn't want to mix things up. Consul has advantages by providing a dedicated service catalog while etcd only has a schema-less key-value store. Registrator is used to register services in Consul together with health checks. It listens to events from the Docker daemon and updates entries in Consul upon container startup and shutdown.

For networking, we chose Weave since it was one of the first to support the network plugin model of Docker and did also work well in our tests. We used the PingPong benchmark of Kratzke [52] to test the candidates and Weave worked best when having enough processing power (multiple CPUs). Project Calico looked very promising from an architectural point of view, but failed in our test installation without any error messages in the logs to deliver larger RESTful messages (>1 kB) in a stable manner. Many packets got lost without any notice. MidoNet is open source on the one hand, but lacks documentation to run it properly on an existing OpenStack cluster. It seems very commercially oriented with an open source offer only to attract attention. Flannel has dependencies on etcd and therefore was no option.

Weave is used in the "one subnet per host" mode, since the advanced modes are not yet supported in the stable version. This led to the need for a reverse proxy to forward requests to the container in case it was migrated to a different host. Nginx was used here due to better experiences of the project team with it than with HAproxy. No formal evaluation of performance was made, since last benchmarks showed HAproxy and Nginx pretty much on par [53]. Starting with version 1.9, Nginx offers TCP load balancing in addition to http load balancing, which was an advantage of HAproxy before and only available in the commercial version of Nginx. Therefore, both seem pretty much on par. Nginx is also used for routing external requests to internal services of the front-end systems.

We didn't find a storage solution that really met our needs. While Flocker is the most mature solution, it

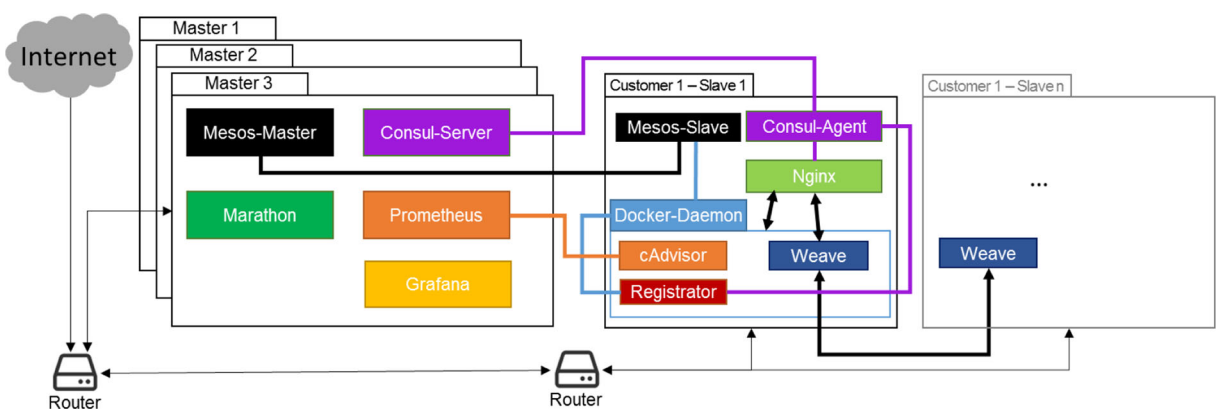


**Fig. 2** overview of cluster management components

brings its own small ecosystem of dependencies that didn't work well with other choices. EMC's REX-Ray reminded us of MidoNet since it is also poorly documented and very commercially oriented so we didn't manage to run it in an open source version. Rancher's Convoy worked as described, but requires two copy operations: one to backup the container volume to a central object store and one to restore it on the new host. That leads to unacceptably long downtimes. The Ceph driver from Yp engineering would suit our purpose best, since it is a small tool without larger dependencies and could be easily integrated. It allows direct usage of block storage from Ceph as a Docker volume, so in case of a container migration it can be unmounted from the old container and remounted on the new container on another host without any copy operation in between. In our scenario however, this would still mean a considerable downtime of a few minutes (1-2) since starting of Nuxeo or Liferay in a container takes that long although it is frequently said that containers start in a few seconds [54, 55]. This is only true for the container itself, but not the application running inside. A suitable solution would be to use snapshots of a volume and mount the snapshot on the new container, while allowing read/write access to the layer above the snapshot on the old container. Once the new container is online, the old container could be shut down and the volume could be remounted with the updated data. However, we have not managed to implement such a solution up to now.

Regarding monitoring, cAdvisor is used for collecting information about container resource usage. This information is stored in a Prometheus database afterwards and can be visualized using a Grafana dashboard. There is no real alternative for cAdvisor. Prometheus was preferred against InfluxDB to store this data, since it claims to store data more effectively. It stores a name and additional labels for every metric only once, whereas InfluxDB stores them redundantly for every timestamp. Both solutions are supported by cAdvisor as an export target. Prometheus does also support alerts (r08). Grafana was chosen over Kibana for visualization since it seems more open towards different purposes whereas Kibana is mainly tailored for the ELK stack. Table 3 provides an overview about the different components in the Kubernetes and Mesos stack. Differences regarding fulfilled requirements to Table 2 result from additional integration code in Kubernetes or our own solution.

**Table 3** Comparison of solution stacks

| Category | Stack 1 | Stack 2 | Reqs. |
|---|---|---|---|
| Orchestration | Kubernetes | Marathon | 4-6 |
| Scheduler | Fleet | Mesos | 10 |
| Master election | Podmaster | Zookeeper | - |
| Service discovery | Etcd | Consul<br>Registrator | 14<br>11, 13 |
| SDN | Flannel | Weave | 15, 16 |
| Storage | Kubernetes | Yp Ceph | 12 |
| Load balancer | Kube-proxy | Nginx | 17,18 |
| Monitoring | cAdvisor<br>Heapster<br>(fluentd) | cAdvisor<br>Prometheus<br>Grafana | 7<br><br>(9) |
| Management | Dashboard | Dashboard | 1-3 |

### 6.4 Evaluation

To highlight the deficiencies of an out-of-the-box solution with the selected components, we performed a number of tests.

1. Create a new virtual machine
2. Migrate a container to a new host
3. Identify a container with high CPU load
4. Compare resource usage for a single application container across customers
5. Vertical scaling of an application based on resource usage

We used a simple application instead of our composite application in order to make the tests simpler and therefore easier to reproduce. The application is a WordPress blog which depends on a MySQL database.

ad 1) *Test*: the VM should automatically contain all necessary components and join the Mesos cluster as a slave. *Procedure*: an OpenStack Heat template or a configuration management system can be used. *Result*: ok. *Discussion*: the solution is external to the Docker management suite and not integrated. It is still an acceptable solution.

ad 2) *Test*: the container should be stopped and restarted on a host with more resources.

*Procedure*: Increase the resource requirements of the application in Marathon. *Result*: a new deployment is created, but it is not carried out. No error visible. *Discussion*: it is a bug in Marathon that is preventing it from shutting down the existing instance. MinimumHealthCapacity is one by default. If set to zero the shutdown works and the container is restarted on a new host. However, it also means that Marathon does not restart the application if it fails.

ad 3) *Test*: the administrator should be able to easily identify a container with high CPU load. *Procedure*: using the Grafana dashboard, look at the CPU graph of the containers. *Result*: identifying the container is easy, but finding out which application it is running and to what customer it belongs is hard. *Discussion*: Mesos assigns the name of the Docker container automatically using the schema mesos<id>. The administrator has to log on to the host and use the Docker commands to find out which application is running.

ad 4) *Test*: a container should be identifiable in the monitoring solution using filters like application name and customer name. *Procedure*: using Grafana, the administrator is filtering the container list. *Result*: not possible. *Discussion*: cAdvisor doesn't deliver any tags which could be used for filtering.

ad 5) *Test*: an additional application instance should be started and load balanced as soon as a high resource usage of the existing instance is detected. If resource usage is low for some time, the additional instance should be shut down. *Procedure*: generate high load using Apache jMeter. Result: no auto scaling possible. *Discussion*: Marathon does not have that feature. A respective request was declined with the remark it would be very customer specific [56].

### 6.5 Own Enhancements

To fix these issues and create a better integrated cluster management solution for Docker, author Florian Pfitzer created a number of small applications and extensions to some of the components used.

*Docker controller* syncs container metadata from the Docker daemon to the Consul service registry. This fixes the problem that information and consul can run out of sync when changes happen on the Docker host while Registrator is not running.

*Marathon scaler* gets statistics for a Marathon application from Prometheus and scales it when needed. It can either create additional application instances with the same resource requirements, or increase the resource requirements of the app and restart it on a host with enough free resources. The blueprint requirement was solved by storing a Marathon JSON file in Consul and pushing it into Marathon with the scaler application (http://bit.ly/1Q8HTiO).

*cAdvisor Pull Request #780* Export image name and environment variables as Prometheus labels. Since Marathon stores the name of the Docker image that is run in the container in an environment variable, exporting it to Prometheus allows filtering for the image name. The same is true for host and Marathon application name.

*Prometheus Pull Request #812* Use Consul ServiceAddress instead of Address when set. This fixes an issue when Registrator is run in internal mode and pushes the IP address of the Docker container to the ServiceAddress field instead of Address. Since version 0.15 of Prometheus this can be also done using a mapping of ServiceAddress to Address in Prometheus.

## 7 Final Considerations

Despite the fact that the Docker ecosystem is huge, there still are requirements not fulfilled by any of the tools (r20, r22) and some are only fulfilled by a single tool (r08, r11). Many tools emerged quite recently and therefore must be considered premature.

Managing customer data (r22) is maybe the most important missing part. [57] 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 and also missing in the OpenStack project, although the latter is about to change with CloudKitty and Cyclops.

Updating a container (r20) can be emulated with a couple of Docker commands replacing it, since containers should be immutable. Still there should be a standard way to automate this and also a solution for possibly necessary one-time tasks that do a database schema update or similar things associated with the update. The rolling-update functionality of Kubernetes and our own solution is not sufficient. Registering a service in the registry is also a neglected requirement (r11). 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. Both storage and networking tools have flourished since Docker's publishing of the plugin system. Storage is handled quite well by Flocker, but seems somehow superfluous with the advent of Docker volume plugins. EMC is already one step further and have shown a working demo of migrating a stateful application container inside a Mesos installation. It's a pity that we didn't manage to get REX-Ray running. EMC really has to work on the documentation. Within the SDN area, Weave worked best in our tests, although we would prefer a solution that directly replaces OpenStack's SDN Open vSwitch and provides a unified solution for both OpenStack and Docker running inside VMs on OpenStack. Project Calico and MidoNet qualify for that. The latter has the same problems like REX-Ray: missing or incomplete documentation. Calico did work for us, but showed problems with delivering larger packets in the Ping-Pong test. Maybe Ciscos engagement in the project will help here, although it is currently unclear whether they want to cooperate as announced or create a fork called contiv which also contains a volume driver with Ceph support. Another candidate that recently got our attention is OpenContrail, an SDN which targets replacing Open vSwitch in OpenStack and recently started supporting Kubernetes [58].

### 7.1 Outlook

In contrast to calls for consolidation [59], the ecosystem is still growing both in existing tool categories as well as in new ones. CRIU (checkpoint/restore in userspace) allows live migration of containers with support of Docker 1.5. Unfortunately it currently has issues with Docker v1.8 and still requires a modified Docker and a modified Linux kernel to work. Although full support in vanilla Docker was announced for version 1.9 [60], it is still not present in 1.10.

Additionally, tools come and go and sometimes developers throw away the old solutions to build something new as ActiveState did with CloudFoundry (CF) BOSH, which is replaced by CF Diego and CF Lattice, which both support Docker, but come with some limitations [61]. Even the underlying Docker engine is challenged with multiple alternatives like rkt from CoreOS, LXD from Canonical and Hyper claim to have better isolation, more security or other benefits.

Although it currently looks like many people focus on either Kubernetes or Mesos/Marathon, there are still new competitors in this area as well.

The Nomad cluster manager was announced by Hashicorp at the end of September 2015 [62]. It is specialized on microservices and Hashicorp claims it is more generally applicable than Kubernetes since it also supports directly running VMs and Java applications in addition to Docker containers and architecturally easier than Mesos with Marathon since it is a single binary with much less external dependencies [63]. It uses Consul and other tools by Hashicorp and targets large clusters across multiple datacenters.

Another recent development is the Microservice infrastructure called Mantl developed at Cisco [64]. It happens to use many of the same components as our own solution, building on Mesos, Marathon and Consul. It uses some different choices in other areas replacing Nginx with HAproxy, Weave with Calico, Prometheus/Grafana with the ELK stack and cAdvisor with CollectD, so that hosts are monitored instead of single containers. They seem to use similar glue code for integrating Mesos/Marthon with Consul as well as HAproxy with Consul. This can be seen as an indication that our approach is going into the right direction.

Hyper has recently announced the Hypernetes project, a combination of OpenStack, the Hyper engine (instead of Docker) and Kubernetes. They claim to have built a true multi-tenant version of Kubernetes and at the same time being more efficient due to elimination of the VM layer [65].

### 7.2 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 [66] 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. Our solution is in line with the reference architecture for lightweight virtualization clusters by Kratzke [67].

Not all requirements are currently fulfilled and despite first integration approaches, there is a need for closer cooperation within the Docker ecosystem. We envision an embracing ecosystem project that bundles the forces of Mesos proponents to jointly work on a holistic solution for the Docker management challenge. With Kubernetes and the upcoming competitors Triton, Nomad and Mantl, there would still be enough competition for a healthy market.

## References


1. Pahl, C.: Containerization and the PaaS Cloud. IEEE Cloud Comput., 24–31 (2015)
2. Scheepers, M.J.: Virtualization and Containerization of Application Infrastructure: A Comparison. Presented at the 21st Twente Student Conference on IT , Twente The Netherlands June 23 (2014)
3. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures-a technology review (2015)
4. Ranjan, R.: The cloud interoperability challenge. IEEE Cloud comput. **1**, 20–24 (2014)
5. Rosen, R.: Linux containers and the future cloud. Linux J **2014**, 3 (2014)
6. Kratzke, N.: Lightweight virtualization cluster how to overcome cloud vendor Lock-In. J. Comput. Commun. **2**, 1–7 (2014)
7. Rubens, P.: Docker No Longer the Only Container Game in Town, http://bit.ly/1IlkI0s (2015)
8. Hecht, L.: How Open Source Communities Power Docker and the Container Ecosystem, http://bit.ly/1LSIoLW (2015)
9. Turnbull, J.: The Docker Book: Containerization is the new virtualization James Turnbull (2014)
10. Lewis, J., Fowler, M.: Microservices, http://bit.ly/1dI7ZJQ (2014)
11. Papazoglou, M.P.: Service-Oriented Computing: Concepts, characteristics and directions. In: Web Information Systems Engineering (WISE 2003). 4Th Int. Conf. On. Pp. 3–12. IEEE (2003)
12. Mietzner, R., Leymann, F., Papazoglou, M.P.: Defining Composite Configurable SaaS Application Packages Using SCA, variability descriptors and multi-tenancy patterns. In: ICIW 2008. IEEE (2008)
13. Cockcroft, A.: State of the Art in Microservices. Presented at the DockerCon Europe 14 , Amsterdam Netherlands December 4 (2014)
14. Evans, E.: Domain Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, Boston (2003)
15. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services. Pp. 527–549. Springer (2014)
16. Roßbach, P.: Docker poster. Entwickler mag docker spez (2014)
17. Docker Ecosystem Mindmap, http://bit.ly/1BjDgtW
18. Peinl, R.: Docker ecosystem on Google Docs, http://bit.ly/1DJ0eS4
19. Docker, Inc.: About, http://bit.ly/1OjEBLl
20. Crane, C.: The Container Ecosystem Project, http://bit.ly/1RkyBTu
21. Wallner, R.: A breakdown of layers and tools within the container and microservices ecosystem, http://bit.ly/21cttZN (2015)
22. Williams, A.: The Docker & Container Ecosystem The New Stack (2015)
23. Chauhan, M.A., Babar, M.A.: Migrating Service-Oriented System to Cloud Computing: an Experience Report. In: Cloud Computing (CLOUD) 2011, IEEE Int. Conf. On. Pp. 404–411. IEEE (2011)
24. Coffey, J., White, L., Wilde, N., Simmons, S.: Locating Software Features in a SOA Composite Application. In: 8Th European Conf. on Web Services (ECOWS 2010). Pp. 99–106. IEEE (2010)
25. Sefraoui, O., Aissaoui, M., Eleuldj, M.: Openstack: toward an open-source solution for cloud computing. Int. J. Comput. Appl. **55**, 38–42 (2012)
26. Koukis, V.: Flexible Storage for HPC Clouds with Archipelago and Ceph. In: 8Th Workshop on Virtualization in High-Performance Cloud Computing. ACM (2013)
27. Chadwick, D.W., Siu, K., Lee, C., Fouillat, Y., Germonville, D.: Adding federated identity management to openstack. J. Grid Comput **12**, 3–27 (2014)
28. Piraghaj, S.F., Dastjerdi, A.V., Calheiros, R.N., Buyya, R.: Efficient Virtual Machine Sizing for Hosting Containers as a Service. In: IEEE World Congress on Services. Pp. 31–38. IEEE (2015)
29. Jain, R., Paul, S.: Network virtualization and software defined networking for cloud computing: a survey. Commun. Mag. IEEE **51**, 24–31 (2013)

30. Day, S.: Docker Registry V2 - A New Model for Image Distribution. In: Docker Con 2015. , San Francisco (2015)
31. Hausenblas, M.: Docker Registries: the Good, the Bad & the Ugly, http://bit.ly/1OsrnIu
32. Mills, K., Filliben, J., Dabrowski, C.: Comparing VM Placement Algorithms for On-Demand Clouds. In: 3Rd Int. Conf. on Cloud Computing Technology and Science (Cloudcom). Pp. 91–98. IEEE (2011)
33. Verma, A., Pedrosa, L., Korupolu, M., Tune, D.O.E., Wilkes, J.: Large-Scale Cluster Management at Google with Borg. In: 10Th European Conference on Computer Systems. P. 18. ACM (2015)
34. Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., Wilkes, J.: Omega: Flexible, Scalable Schedulers for Large Compute Clusters. In: 8Th ACM European Conf. on Computer Systems. Pp. 351–364. 1 (2013)
35. Bucchiarone, A., Gnesi, S.: A Survey on Services Composition Languages and Models. In: Intl. Workshop on Web Services–Modeling and Testing (WS-Mate 2006). P. 51 (2006)
36. Caballer, M., Blanquer, I., Moltó, G., de Alfonso, C.: Dynamic management of virtual infrastructures. J. Grid Comput **13**, 53–70 (2015)
37. Bachlechner, D., Siorpaes, K., Fensel, D., Toma, I.: Web Service Discovery-A Reality Check. In: 3Rd European Semantic Web Conference (2006)
38. Vukojevic-Haupt, K., Haupt, F., Karastoyanova, D., Leymann, F.: Service Selection for On-demand Provisioned Services. In: 18Th Intl. Enterprise Distributed Object Computing Conference (EDOC'14). Pp. 120–127. IEEE (2014)
39. Serebryany, I., Rhoads, M.: SmartStack: Service Discovery in the Cloud, http://bit.ly/1bRcjo2 (2013)
40. Swan, C.: ClusterHQ Launches Flocker to Facilitate Robust Stateful Docker Containers, http://bit.ly/1KH3zG3 (2014)
41. Hall, S.: Five Storage Companies That Speak To Docker's Next Wave, http://bit.ly/1VBHYzp (2015)
42. Han, S.: Getting Started With the Docker RBD Volume Plugin, http://bit.ly/1RYVONi (2015)
43. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput **12**, 559–592 (2014)
44. Costache, C., Machidon, O., Mladin, A., Sandu, F., Bocu, R.: Software-Defined Networking of Linux Containers. In: 13Th Roedunet Conference. IEEE (2014)
45. Drutskoy, D., Keller, E., Rexford, J.: Scalable network virtualization in software-defined networks. IEEE Internet Comput. **17**, 20–27 (2013)
46. Rimal, B.P., Jukan, A., Katsaros, D., Goeleven, Y.: Architectural requirements for cloud computing systems: an enterprise cloud approach. J. Grid Comput **9**, 3–26 (2011)
47. Liu, H., Wee, S.: Web Server Farm in the Cloud: Performance Evaluation and Dynamic Architecture. In: Cloud Computing. Pp. 369–380. Springer (2009)
48. Aceto, G., Botta, A., De Donato, W., Pescapè, A.: Cloud monitoring: a survey. Comput. Netw **57**, 2093–2115 (2013)
49. Ward, J.S., Barker, A.: Observing the clouds: a survey and taxonomy of cloud monitoring. J. Cloud Comput. Adv. Syst. Appl. **3**, 40 (2014)
50. Yegulalp, S.: Docker Datacenter promises end-to-end container control for enterprises, http://bit.ly/1QYXoK1 (2016)
51. Polvi, A.: The Security-minded Container Engine by CoreOS: rkt Hits 1.0, http://bit.ly/1S3iyw5 (2016)
52. Kratzke, N.: About Microservices, Containers and their Underestimated Impact on Network Performance. CLOUD Comput. **2015**, 180 (2015)
53. Kazemier, A.: BalancerBattle, https://github.com/observing/balancerbattle
54. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An upyeard performance comparison of virtual machines and linux containers. Research Report RC25482, IBM Almaden (2014)
55. Seo, K.-T., Hwang, H.-S., Moon, I.-Y., Kwon, O.-Y., Kim, B.-J.: Performance comparison analysis of linux container and virtual machine for building cloud. Adv. Sci. Technol. Lett. Netw. Commun **66**, 105–107 (2014)
56. Kabhal: Introduce multiple scaling strategies mesosphere/marathon Issue #1477, http://bit.ly/1QAETBl (2015)
57. Lindner, M., Galán, F., Chapman, C., Clayman, S., Henriksson, D., Elmroth, E.: The Cloud Supply Chain: a Framework for Information, Monitoring, Accounting and Billing. 2Nd Int. Conf. on Cloud Comp (2010)
58. Sreelakshmi, S.: OpenContrail – Kubernetes Integration, http://bit.ly/1Q91Jun (2015)
59. Peinl, R., Holzschuher, F.: The Docker Ecosystem Needs Consolidation. In: 5Th Intl. Conf. on Cloud Computing and Services Science (CLOSER 2015) 535-542 SCITEPRESS, Lisbon, Portugal (2015)
60. Kazemi, S.: CRIU Support in Docker for Native Checkpoint and Restore. In: Linux Plumbers Conference 2015. , Seattle, Washington, USA (2015)
61. Berman, L.: Are Diego and Docker Really Good Friends?, http://bit.ly/1WGpAW4 (2015)
62. Dadgar, A.: Nomad, http://bit.ly/1MV4bYB (2015)
63. Hashicorp: Nomad vs. Other Software, http://bit.ly/1OsqLCG (2015)
64. Owens, K.: Building Cisco's IoE PaaS with Mantl, http://bit.ly/1KFP9Ck (2015)
65. Yegulalp, S.: Hypernetes unites Kubernetes, OpenStack for multitenant container management, http://bit.ly/1QvVrUS (2015)
66. Crisp Research Open cloud alliance - openness as an imperative crisp research (2014)
67. Kratzke, N.: A lightweight virtualization cluster reference architecture derived from open source PaaS platforms. Open J. Mob. Comput. Cloud Comput **1**, 17–30 (2014)