2015 IEEE Conference on Open Systems (ICOS), August 24-26, 2015, Melaka, Malaysia

Evaluation of Docker as Edge Computing Platform

Bukhary Ikhwan Ismail, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing Yuan Luke, Ong Hong Hoe
Advanced Computing Lab
MIMOS Berhad
Kuala Lumpur, Malaysia
{ikhwan.ismail; ehsan.goortani; bazli.abkarim, mt.wong; sharipah; jyluke; hh.ong }@mimos.my

***Abstract*—High latency, network congestion and network bottleneck are some of problems in cloud computing. Moving from centralized to decentralized paradigm, Edge computing could offload the processing to the edge which indirectly reduces application response time and improves overall user experience. This paper evaluate Docker, a container based technology as a platform for Edge Computing. 4 fundamental criteria were evaluated 1) deployment and termination, 2) resource & service management, 3) fault tolerance and 4) caching. Based on our evaluation and experiment Docker provides fast deployment, small footprint and good performance which make it potentially a viable Edge Computing platform.**

***Keywords*—Edge Computing, Mobile Cloud, Mobile Edge Computing, Cloudlet, Fog Computing; Container; Docker; simulation.**

I. Introduction

The number of mobile devices accessing applications and services from the cloud increases. Using these services from cloud creates high latency to application, congestion and bottleneck to the network. Edge computing solution tries to minimize the impact by processing application and data closer to the user. One way to take advantage of Edge Computing (EC) is to exploit the use of Linux containers to host applications at the edge.

Linux containers offer a lightweight, portable and high performance alternative to virtualization. The size of container image which is smaller than Virtual Machine (VM) images makes it suitable to launch applications at the edge faster than VM based appliances.

There are two modes of deploying Linux containers. LXC offers OS containers that are lightweight and behaves similar to a VM. User may login to system and install its own application. Another option is Docker, another container technology where it is designed to run a single application per container. Unlike LXC, Docker is known with its focus to build loosely coupled applications.

This paper describes our evaluation of Docker as the EC platform. Our evaluations will be based on four fundamental requirement features of EC should have; 1) deployment and termination; 2) resource & service management; 3) fault tolerance and 4) caching. This paper is organized as follows. Section II, discuss the background of EC and differences between the Cloud and EC paradigm. Section III discuss on the related work. Section IV describes the Docker technology for EC platform. Section V and VI describes the use cases and our technical evaluation of Docker. Section VII, discuss our observation and finally we conclude in Section VIII.

II. Background

A. Edge Computing

Edge computing shifts the paradigm from centralized to decentralized; by utilizing compute, network and storage resources that is closer to the user. It pushes the content and service away from the center nodes e.g. Datacenter (DC) or cloud to the logical extremes of a network. Ideally a “one hop” away from the user, the closer the application and content, the better Quality of Experience (QoE) user attain [1]. By eliminating or deemphasizing on centralized environment it removes bottlenecks and potential point of failure, thus making it more preferably resilient to failure.

EC aims at reducing response time or latency, by caching or offloading content to the edges. By running from the edges, it creates potential new service categories and new unexplored business models [2]. Some of potential new services are location based, Internet of Things (IoT) , data caching, big data, sensor monitoring activities e.g. air conditioning, elevators, temperature, humidity, retail solutions or public safety applications [3].

EC may have different names. Some of the term are Fog Computing [4], Virtual Cloudlet [5], Mobile Cloud and others. Some may argue the similarities and dissimilarities of these technologies, but in term of the objective it is almost the same, where the resources are setup closer to the user in order to increase the user’s QoE.

B. From Centralized to Decentralized; Cloud to Edge Paradigm

Cloud computing has reached its maturity. It serves as a good business and user model. Low startup cost, pay per use, on-demand and elasticity of resources put Cloud computing on an attractive proposition for businesses. From user perspective, the application and data can be reached from anywhere, everywhere and anytime.

Cloud with its maturity and commercial advancement has its disadvantages. Some of the limitation are WAN latency, bandwidth and application response time. Data and applications are processed in the cloud where every bit of data is transferred over the network. It is time consuming especially for large data sets to travel back and forth between user and cloud [3]. Table 1 shows the advantages of EC over cloud computing.

TABLE I: ADVANTAGES OF CLOUD COMPUTING VS. EDGE COMPUTING

Requirements	Cloud Computing	Edge Computing
Latency	High	Low
Delay Jitter	High	Very low
Location of service	Within the Internet	At the edge
Distance client and server	Multiple hops	One hop
Location awareness	No	Yes
Geo-distribution	Centralized	Distributed
Support mobility	Limited	Supported
Real time interaction	Supported	Supported

* Taken from CISCO Blog. [3]

To bring application and data closer to the user, there are 3 variance of EC infrastructure setup. In *Mobile Edge Computing* (MEC), the caching and application resides in the Radio Access Network (RAN) of service provide infrastructure or mobile operators facilities [6]. In an *intermediate setup*, a pool of resources within the vicinity of the company or organization can sit between DC and user [7]. While some regard "closer to user" as within the reach of its neighboring devices i.e. by forming ad-hoc mesh network of resources to commonly serve all user or devices needs [8].

III. RELATED WORK

A Cloudlet project which runs an Augmented Reality application uses an *Execution Environment Platform* adopted from OSGi. OSGi is a service oriented module management system based on Java. The choice of Java is simply because it is common for hardware and OS platform to support it. OSGi allows developers to load and unload software modules called *bundles* dynamically during runtime [5].

Other project proposed a VM based Cloudlet infrastructure [7] [9]. It emphasis is on *transient state* application, that allows the infrastructure to host applications on-demand and perform rapid teardown afterward. There is not effect on the underlying server that host the VM. The VM encapsulates all the applications, libraries and data thus making it a suitable for multi-tenant applications to run on this environment.

ETSI in 2014 forms a new Industry Specification Group to create industry specification for MEC. The goal is to create a sustainable business for all players in the value chain by proposing standardization and open environment across multi-vendor e.g. mobile operators, equipment and application platform vendors. Applications can exploit the ultra-low latency and high-bandwidth within the infrastructure. MEC can be seen as a cloud server running at the edge of a mobile network [2] [6].

PeerApp mobile solution brings optimization functionality such as content and DNS caching to RAN. It allows applications and a service run closely to the subscribers by reducing the number of hops user need to pass which indirectly increases user's QoE [10].

Kaval is an Android based platform provides *signature based identification* facility. Kaval enabled devices connect to its neighboring peers to form a mesh network. It will self-organize to cooperatively capture and analyze audio or images. It improves the signature process response time by sharing data and distributing the workload among its peers [1] [8].

BOINC is a voluntary computing platform that runs on Windows, Mac, Linux or Android devices. Scientists, universities or companies can create a computing power of a thousand CPUs. The use of BOINC across multiple hardware and OS platform makes it a good tool to run scientific research application such finding a cure to disease [11].

HTC Power to Give is another mobile application that leverages on mobile devices as compute platform. Each devices act as a middleware to connect to research projects based where user able choose research application to run. Research problem, are broken down into many small task and distributes across multiple devices. Once complete, results is push back to centralized server [12].

IV. DOCKER AS EDGE COMPUTING PLATFORM

In this section, we explain in detail how Docker functions.

A. Container Technology

Cloud providers may host Platform as a Service (PaaS) thru virtual machine or container based technology. In 1979, UNIX introduced *Chroot* command as part of UNIX version 7. Later in 1988, FreeBSD introduce *jail* that extends from *Chroot*. Sun Solaris 10 introduces *Zones* which extend the capabilities of *Chroot*. In Sun Solaris 11, *Zones* extends to become *Containers*. Later, Linux introduce LXC. LXC use *userspace* interface for the Linux Kernel containment functionality. User are able to manage LXC through API and CLI tools [13].

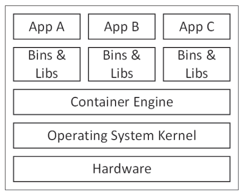


Fig. 1. Container Architecture

Fig. 1 shows the components in container technology. Each application consists of the application, binaries and libraries that are packed as an independent container. Each container is isolated and consists of its own independent subsystem of network, memory and file system. The *Container Engine* manages those containers [14]. Since the *Container Engine* extent from Operating System (OS) Kernel, containers on the same *Container Engine* share the same OS in a fraction of the size. This allows hundreds of containers run within a single OS. It gives container the benefit of rapid deployment with near-native performance in CPU, memory, disk and network.

B. *Docker as a candidate*

Docker has two main use cases i.e *continuous integration* and *continuous deployment*. With Docker being lightweight, developers can build stacks of Docker containers on their laptop that replicate some of production environment. In our paper we evaluate Docker for an alternative objective; for EC platform.

Docker manage container by managing *namespaces.* Docker uses five different *namespaces*, they are; *Process ID* (PID), *Networking*, *Inter-Process Communication* (IPC), *Mount* (MNT) namespace and *UNIX Timesharing System* (UTS). Docker use *control group* (cgroups) to manage available hardware resources among containers. It uses a lightweight file system called *UnionFS* to provide the building blocks for containers. All these components are combined into a format called *libcontainer* [15].

V. Edge computing requirements

This paper evaluates Docker containerized environment for EC with the following features:-

A. *Deployment and Termination*

Environment at the edges may consists of low end devices that are not comparable e.g. performance or capacity to servers in DC. Personal PC, laptops, mobile devices are some of common compute power at the edges. Geo-distributed computation at edges requires a platform to be flexible enough to handle application or service deployment and management. The flexibility of candidate platform i.e. Docker requires it to be able to deploy reusable service without dependency on heterogeneous devices. Edge devices should be ideally manage from both; centralized i.e. DC and within the edge itself i.e. devices at the edge. The platform should provide an easy way to install, configure, manage, upgrade and termination of running services [16]. Termination of services must not affect other services or resources within the edge and the free resources may be utilized by other services within the edge.

B. *Resource and Service Management; Service Registry, Resource Join and Leave the Edge*

To enable efficient management of applications at the edge, we need *service registry* and *service discovery*. Edge environment is an uncontrolled environment, devices are heterogeneous and these resources might come and go. For example, if there are no resources at one edge, the services should be deployed at the neighboring edge where resources exist. On the other hand, if the resources exist with very minimum specifications, the service should launch but users would experience certain levels of degradation. If however a new resource appears in the edge, the user should experience certain level of progressive improvement. Join and leave edge cluster is also an important feature to ensure the service can still serving the user even when the resources are leaving.

C. *Fault Tolerance*

The choice of platform must be fault tolerant ready in order to achieve high availability and reliability of the infrastructure. High availability of platform is an important feature where data and services reside on edges rather than centralized location. A study by D. LeClair suggested that EC availability should surpass the typical three nines (99.9%) of data center. It suggest EC should support at least five (99.999%) availability on the edge [16].

D. *Caching*

Caching can be performed at the edges to decrease user local access and to improve overall application performance. One scenario is that Docker images can be cached at the edge to reduce the application provisioning time. Another angle would be to support application data cache in order to improve application response time.

VI. The Experiment

In order to examine Docker as one of the candidate technology to enable Edge computing, a testbed have been setup as show at Fig. 2 The testbed consists of a datacenter and three edge sites to simulate the environment.

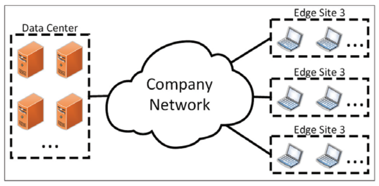


Fig. 2. Network Topology

At each edge site, a *Docker Registry* is setup to store *Docker images* locally at the edge. *Docker Daemon* at edge site will be able to search and pull the *Docker Image* from the *Docker registry*. *Docker Swarm is* configured on each edge site to manage multiple *Docker Daemons*. *Docker Swarm* acts as a clustering and orchestration tool. *Client* e.g. CLI or web-based portal through Docker Shipyard will be able access and control the Docker environment.

Each edge site will have different properties e.g. different number of hosts per edge, heterogeneity of devices where both criteria tries to simulate the environment in EC. The network between datacenter and edge sites will simulate the WAN latency in EC.

A. *Deployment and Termination*

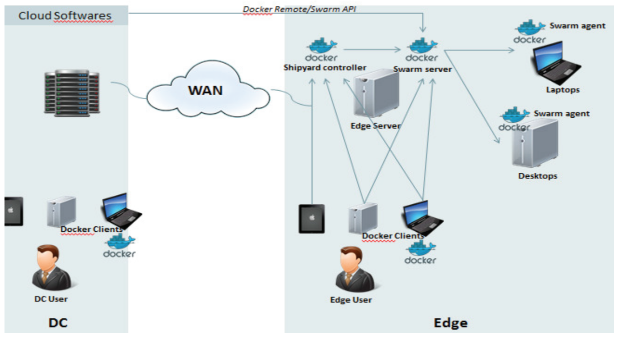


Fig. 3. Service deployment with Docker Shipyard and Docker Swarm

In this experiment, we deploy application in a container at the edges. Fig. 3 show the setup, a DC and an single edge site configuration. Container can be deployed from DC or within the edge cluster. By default Docker allows remote deployment of container through an API, i.e. via *Shipyard WebUI* or *Docker Client* that communicate directly to *Docker Swarm.*

To setup an application, we configure the *Dockerfile* to include necessary *commands* to build the image. After building the image, it is pushed to the local registry at the edge. By pushing images from datacenter to the edge local registry we are able pre-cached the images which indirectly reduce the network usage and decrease service deployment time.

Client submit request for a new container to the *Swarm Manager*. To find the most suitable Docker daemon, *Swarm Manager* performs scheduling and filtering. The request is then submitted to the selected *Docker Daemon*. As long as the image is available locally, Docker deploys the service as a container in the edge cluster.

Docker applies layering concept to Docker image[17]. It only commits the delta or data differences on to the image making the image size significantly reduced. Image reusability is also provided via local *Docker Registry* or *DockerHub*. These components allow fast deployment of any services at anywhere we want.

Manageability of Docker Daemon and container can be performed through *Shipyard*. application is not easily manage in a distributed edge locations settings. Management can be performed at each separate edge locally, but ideally in EC environment, it must be able to oversee or control the overall EC clusters through from example DC. To manage multiple edge sites into a single cluster pool would be challenging. Currently, Docker does not support hierarchal or distributed management either out of the box or through any other compatible components.

For service termination, Docker recently added a feature to automatically clean up container once it is no longer needed. Each deployment of containers may cause resource wastage from remaining and unwanted containers. This feature should not be use while data volumes are attached to the container. A graceful way of handling detaching the data volumes should be use to reduce the risk of data loss.

B. *Resource and Service Management; Service Registry, Resource Join and Leave the Edge*

Docker Swarm supports service registry and discovery in order to manage services in containers. There are many backend models that Docker Swarm can support right now; *hosted discovery* and *Consul* [17], *etcd* [18] and *zookeeper* [19] and each of them has different implementation requirement to meet.

Fig.4 described the setup of Docker Swarm cluster with Consul. Consul was evaluated because it can provide more cluster information compared to hosted discovery. Fig. 5 describes the states of Docker engines when one of the nodes is down. The leaving Docker daemon will trigger a graceful leave command and shutdown the Consul daemon. This is used to ensure other nodes see it as "*leave*" instead of "*failed*" where node that leave will not attempt to re-join the cluster.

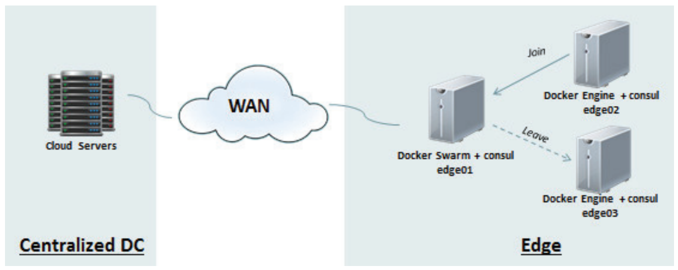
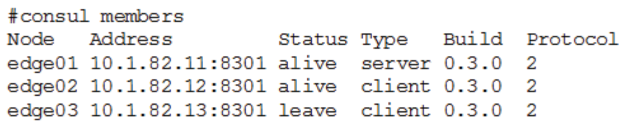

Cloud Servers
WAN
Join
Docker Engine + consul edge02
Leave
Docker Swarm + consul edge01
Docker Engine + consul edge03
Centralized DC
Edge

Fig.4. Edge Environment Setup of Docker Swarm and Consul as service discovery backend



```
#consul members
Node   Address         Status Type   Build  Protocol
edge01 10.1.82.11:8301 alive  server 0.3.0  2
edge02 10.1.82.12:8301 alive  client 0.3.0  2
edge03 10.1.82.13:8301 leave  client 0.3.0  2
```

Fig. 5. Consul command to check node join and leave the cluster

C. *Fault Tolerance*

Check-pointing of Service - CRIU is an open source project designed to checkpoint and restore Docker container. It checkpoint the states (network, memory, CPU, transaction) and restore it back for recovery purposes. It can be use in a situation where the application does not support state-less transaction.

Volume Management - Flocker is a container data volume manager. It allows state-full container e.g. production database data to be protected. Flocker data volume called *dataset* is portable. It can be use by any container regardless of the location of the container running. There are two options to use Flocker. First, Flocker manages both Docker Containers and Data volumes together where volumes follow the container when it moves to different hosts. Alternatively it can manage separately the data volume and container.

Offline migration - In our experiment, we used Docker features to migrate containers. Using Docker export/import features we are able to export container to a TAR file, and import it to different Docker daemon at different host. Using export/import, does not preserved the memory nor CPU instruction as it is not the same as live-migration in hypervisor based virtualization e.g. KVM, Xen. The application within the container must be stateless and loosely coupled with any interacting components. In an event of failure, the container can be provision back without affecting the overall system availability.

Multiple zones - By having distributed edge zone across geographical site, data and application can be placed strategically at multiple zones. Failover, disaster recovery, load balancing mechanism can be setup between the DC-to-edges or edge-to-edge site. It can alleviate the point of failure which indirectly increase the availability of application and data availability [4].

D. *Caching*

Ideally data collection, normalization and filtering should be done at the edges before data is push to for example database at the DC. This will reduce the total data being pushed to the DC. Caching data at the edges helps improving performance as well as enhance capability of failure recovery. We foresee two types of data can be cached. We need to cache the application Docker image and application data at separate edges to provide faster reach to the user.

Caching data in the containerized platform can be performed by caching containers data volume. By creating a shared space for container's volumes, these data volumes are dynamically collected at the edges. For example, in a data analytics platform, application in Docker can be configured to use shared data volumes. There are also variety of methods to enhance Edge Computing faster for data access and load distribution, for example caching data via data replication at the edges [20]. In Docker platform, caching data via shared and replicated volumes offers decoupling data from services.

E. *Applications*

Hadoop is a framework for processing large data in parallel. Hadoop developed in Java and support any programming language with "Hadoop Streaming". Hadoop consists of two main layer which are *MapReduce layer* and *Hadoop Distributed File System* (HDFS) layer. MapReduce layer consist of two phases which are *map phase* and *reduce phase*. In map phase process the data based on the logic in the map() function. The output from the map() function then become the input for the reduce phase. In reduce phase organize the data based on the logic in the reduce() function.

We tested Hadoop by deploying it on our testbed. During the experiment, Docker give advantages in installation. We configure Hadoop in Docker using Docker Ferry. Docker Ferry simplifies the setup time and reduces configuration errors.

VII. Observation & discussion

Docker Design Concept and Performance - Docker is designed for *single-app per-container* basis and loosely coupled between Docker Container to another. Docker is a platform with low compute footprint. Compare Docker with VM running on hypervisor, hypervisor occupies about 10 to 15 % of host resources but Docker use minimally the host resources [21]. Overall, CPU, memory, storage and network performance is similar to bare-metal [22]. Since Docker container does not virtualize hardware, Docker container is much lightweight and fast. In average, a Docker container run 26X faster than a VM [23]. The overhead of hypervisor is overwhelming and the overhead becomes exponentially greater when more VMs running within the same machine [24]. Docker container can run on small device to large server making it an attractive compute platform to run on edge servers where edge might have lower resource capacity compared to DC.

Docker Agility - Docker images are small and lightweight. It makes Docker agile, portable and easily to transport [23]. After configuring the application in the Docker container, the Docker container is easy to shift from one location to another. This is a good trait for EC where application or service can be shifted to the edge closer to the user with lower data transfer overhead.

Low Storage Footprint - Data is non-persistence in Docker container. In order to save changes in the Docker container, admin needs to commit those changes into the Docker image. Docker will then create additional layer on top of existing image. For each additional layer, it stores only the delta or changes. In a scenario where a single host consists of multiple applications on multiple flavors of OS distro e.g. MySQL in Redhat Linux, PHP in Ubuntu Linux and Application server in Centos Linux. Docker container package up all the required libraries of a distro for all applications. When multiple Docker container shares the same host, only the differences between containers are stored in the storage. Docker contribute to better storage saving and require less storage space where storage is scares at the edge.

Fast Service Deployment and Teardown – One of the important EC characteristic is able to provisioning and teardown service fast. With Docker images being small, this allows the Docker image copy and deploy in a very short time. Since the Docker container does not have a boot up process, the application process within container is able to start instantaneously. When the application is no longer needed, the container can be tear down and no traces will be left on the host OS. This Docker property is suitable for EC environment where multiple application requests can serve, spawn and teardown to make room other user requests [21][22].

Edge Server Location - In MEC, high density server is setup at the Service Provider (SP) infrastructure to host multiple Virtual Machines [6] [5]. Each VM hosts multiple applications and data cached. For companies, local edge server can be setup up to serve as a caching or hosting application for the user. Both MEC and local edge server setups reduce the number of hop for user to reach the application and data. Docker container can replace VM based EC or Cloudlet with lower footprint. Docker container can setup and configured in the extreme edges of the network and within the user physical space e.g. application and data resides on neighboring devices. In another example, Docker container may be a part of the corporate infrastructure environment as a trusted or purely voluntary computing basis. Both use cases are suitable to use container based technology.

VIII. Conclusion

In this paper we have conducted technical evaluation and experiment on selected fundamental Edge Computing use cases. These use cases are in our opinion is the basic building block for EC platform. From the evaluation, we conclude that Docker is a viable candidate. Even there are some challenges; there are still rooms for improvement. Overall Docker provides fast deployment, elasticity and good performance over VM based EC platform. It makes Docker much attractive technology as compared to virtualization based EC technology.

References

[1] R. LaMothe, “Edge computing,” Washington, 2013.

[2] European Telecommunications Standards Institute (ETSI), “Executive Briefing – Mobile Edge Computing (MEC) Initiative,” Sophia Antipolis, 2014.

[3] M. Abdelshkour, “IoT, from Cloud to Fog Computing,” 2015. [Online]. Available: http://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing. [Accessed: 26-Jun-2015].

[4] T. H. Luan, L. Gao, Y. Xiang, Z. Li, and L. Sun, “Fog Computing: Focusing on Mobile Users at the Edge,” in *arXiv preprint arXiv:1502.01815*, 2015.

[5] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, “Cloudlets,” in *Proceedings of the third ACM workshop on Mobile cloud computing and services - MCS ’12*, 2012, pp. 29–36.

[6] H. M. Patel, Y. Hu, P. Hédé, I. B. M. J. Joubert, C. Thornton, B. Naughton, I. Julian, R. Ramos, C. Chan, V. Young, S. J. Tan, and D. Lynch, “Mobile-Edge Computing— Introductory Technical White Paper,” Sophia Antipolis, 2014.

[7] N. Fernando, S. W. Loke, and W. Rahayu, “Mobile cloud computing: A survey,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, 2013.

[8] M. J. Henry, “Kaval : Cooperative Signature Identification on Mobile Devices,” 2012. [Online]. Available: vis.pnnl.gov/pdf/fliers/Kaval.pdf. [Accessed: 25-Jun-2015].

[9] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The Case for VM-Base Cloudlets in Mobile Computing,” *Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, 2009.

[10] PeerApp, “PeerApp Joins ETSI and its Mobile Edge Computing Group,” 2015. [Online]. Available: http://www.peerapp.com/press/peerapp-joins-etsi-and-its-mobile-edge-computing-group/. [Accessed: 26-Jun-2015].

[11] University of California Berkeley, “Open-source software for volunteer computing and grid computing.” [Online]. Available: https://boinc.berkeley.edu/. [Accessed: 26-Jun-2015].

[12] HTC, “HTC Power To Give,” 2015. [Online]. Available: http://www.htc.com/us/go/power-to-give/. [Accessed: 26-Jun-2015].

[13] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, 2014.

[14] M. G. Xavier, I. C. De Oliveira, F. D. Rossi, R. D. Dos Passos, K. J. Matteussi, and C. a. F. De Rose, “A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds,” *2015 23rd Euromicro Int. Conf. Parallel, Distrib. Network-Based Process.*, no. FEBRUARY, pp. 253–260, 2015.

[15] D. Liu and Z. Libin, “The research and implementation of cloud computing platform based on docker,” in *11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2014, pp. 475 – 478.

[16] D. LeClair, “The Edge of Computing: It’s Not All About the Cloud,” 2014. [Online]. Available: http://insights.wired.com/profiles/blogs/the-edge-of-computing-it-s-not-all-about-the-cloud#axzz3TP63LmZw. [Accessed: 25-Jun-2015].

[17] “Consul.io.” [Online]. Available: consul.io. [Accessed: 24-Jun-2015].

[18] “etcd.” [Online]. Available: https://coreos.com/etcd/. [Accessed: 26-Jun-2015].

[19] “Zookeeper.” [Online]. Available: https://zookeeper.apache.org. [Accessed: 25-Jun-2015].

[20] Yi Lin, B. Kemme, M. Patino-Martinez, and R. Jimenez-Peris, “Enhancing Edge Computing with Database Replication,” in *Reliable Distributed Systems, 2007*, 2007.

[21] C. Anderson, “Software Engineering;Docker,” in *IEEE Software*, 2015, pp. 102–105.

[22] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An Updated Performance Comparison of Virtual Machines and Linux Containers,” *Technology*, vol. 25482, 2014.

[23] C. Anderson, “Docker, Software Engineering,” in *IEEE Software*, 2015, vol. 32, no. 3, pp. 102–105.

[24] B. I. Ismail, D. Jagadisan, and M. F. Khalid, “Determining Overhead, Variance & Isolation Metrics in Virtualization for IaaS Cloud,” in *Data Driven e-Science*, 2011, pp. 315–330.