The applicability of container technologies for building a secure production ready PaaS

Ruben Vasconcelos, February 8th, 2018. *2nd year, M.Sc. in Computing.*
Student Number: 16212630, email: ruben.vasconcelos3@mail.dcu.ie

***Abstract*—The way we deliver and manage production software has drastically changed over time. In order to cope with shorter delivery periods in a highly competitive market, companies started investing their efforts into building their PaaS capabilities. Adopting some of the latest technology such as Docker is proving troublesome for smaller IT companies. In this paper, we will examine the main challenges companies are faced with and review some of the related literature.**

Keywords*—*Cloud Computing, PaaS, Containers, Orchestration, Logging, Monitoring, Continuous Integration (CI) and Continuous Delivery (CD).

I. INTRODUCTION

The way Software companies deliver and manage production software has drastically changed over time. Traditionally applications ran directly on hardware; in order to deal with periods of higher demand, companies had to acquire extra hardware and often manually configure it. During the times of lower demand, hardware just sat there underused. Poor hardware utilization induced other secondary effects such as wasted energy and poor software level security. These secondary effects were often caused by a drift in configuration over time; this is a very well-known problem known as server sprawl [1].

A. Virtual Machines

Virtualization techniques such as Virtual Machines (VMs) and Virtual Storage, were initially used as a way of ensuring server consolidation; they helped solve many of the traditional issues associated with running software on bare-metal. However, these technologies would become the base for what we today know as cloud computing [2], because of their inherent elasticity, that allowed for rapid scaling of resources in a way that non-virtualized environments found hard to achieve. Virtualization had a major impact on the software delivery industry; companies like Amazon and Microsoft heavily invested in the creation of IaaS (Infrastructure as a Service) systems, used by many software companies to provision virtual resources on demand.

VMs come with some overheads; they tend to be quite large in size and the guest OS requires extra resources. In addition, the lack of Standardization caused vendor lock-in, where companies spent time and effort automating/adapting their delivery process to a specific vendor and found it difficult to change afterwards.

B. Containers

The next step in the evolution of virtualization is OS level virtualization "Containers" [3]. Containers make it very easy to deploy applications as services across multiple servers/VMs and even different IaaS. Containers share an operating system, have a minimal start-up/shutdown cost, and a very low resource usage in comparison to VMs, helping to reduce overhead and dependency issues [4].

Docker is one of the world leaders in the delivery of container technologies [5].

Key concepts:

- A Dockerfile is a text document that contains all the commands a user could call upon to assemble an image.
- A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. The modular design supported by Docker images enables users to make use of existing base images, which can then be extended by separate Docker images for particular projects. Re-using existing images reduces the effort required to set up an environment, contributes to the environment standardization within a field, and makes great use of the ability of Docker's distribution system to download only differences [6].
- Containers are running instances of an image.
- The DockerHub is a public central image repository that can be used to store images.

While containerizing a new application using Docker is relatively straightforward, running these in production is a totally different matter, typically requiring numerous iterations and long periods of time for converting traditional software delivery pipelines and environments.

II. EVOLUTION OF PAAS

According to Claus Pahl from the Irish Centre for Cloud Computing and Commerce, containers had a direct impact on the evolution of PaaS (Platform as a Service). Many companies are now building their own sort of internal PaaS that they deploy on internal data centres or on public IaaS clouds. Pahl breaks the evolution of PaaS into 3 Generations [7]:

- **First generation:** Consisted of classical fixed proprietary platforms such as Azure or Heroku.
- **Second generation:** Built around open source solutions, which allowed users to run their own PaaS on-premises or in the cloud, with some of these already built around containers.
- **Third generation:** Composed of platforms such as Dawn, Deis, Flynn, Octohost, and Tsuru, which are

built on Docker from scratch and are deployable on a company's own servers or on public IaaS clouds.

Dr Daniel Beimborn and Stefan Wenzel's definition of PaaS states that "PaaS offers a complete set of technologies which are required to develop and to operate SaaS applications. Additionally, many platforms provide marketing and sales opportunities (e.g., Google App Marketplace) and other services along the software value chain" [8]. The third generation lacks some of the fundamental services for managing production applications, that are covered further in this paper such as logging, monitoring and user management.

So it can be argued that the third generation PaaS tools outlined by Pahl are in fact not full production-ready solutions, but rather a tool to be used in the implementation of PaaS. Unlike some of the other generations. For example, Heroku might not natively provide a solution for every issue, but makes use of an extensive list of official plugins and build packs for providing the missing services. Setting up these plugins is rather trivial in comparison to the work required to configure and extend the functionality of the third generation.

Vendor lock-in is still an issue, although it has been passed down to the underlying layers. Containerized applications using Docker are portable across different Docker-based platforms, but the platform developers and DevOps teams regularly have to develop very complex delivery systems, that are directly tied to the underlying infrastructure making it very vendor dependent. Due to fear of vendor lock-in or because of specific business requirements, companies will continue to work on developing their own container platforms, by using other technologies or implementing their solutions to address many of challenges associated with running containers in production which are covered some in more detail later.

III. Reasons behind the interest in container technologies

In order to understand why companies are willing to invest vast resources in such technologies, one must first identify and understand the delivery challenges that software development companies are currently facing. The success of software development organizations is often directly linked to their ability to deliver software fast and reliably. According to Paul Clarke, "Software releases would occur one or two times per year, however, with current competitive market opportunities this has been reduced to weekly, daily and hourly time periods". Putting organizations under more pressure than ever before [9].

One of the solutions found by companies to face this challenge is the use of Open Source tooling for implementing complex delivery pipelines that help achieve a faster delivery of software. NearForm is an example of a company successfully using container technologies. This example is described by P. Clarke in "Technology Enabled Continuous Software Development", where these technologies have highly impacted their software quality and delivery processes [9].

The author describes the components driving NearForm's delivery process. Out of these the following two are directly related to container technologies:

- Continuous deployment model, layered over the Docker container engine, whereby services may be deployed without perturbing the system as a whole.
- Distributed revision control and source code management helps ensure quality, through steps such as code commit hooks via GitHub and the Travis CI toolset.

IV. Challenges

Many large IT companies that developed their own custom solutions to address their specific business requirements are successfully using these technologies in production. But the amount of work and specialized knowledge to implement them is far too great, which in addition to the lack of standardization is making it difficult for new small IT companies to employ similar approaches [10]. In order to better our understanding of how to successfully implement similar concepts, one must first understand the challenges in addition to the number of resources and time required on the long journey to building a production-ready container based PaaS. So we will take a deeper look at some of the challenges associated with building a container PaaS.

A. Cloud Infrastructure

One must first decide on the underplaying infrastructure, a presently three types stand out; public cloud, private cloud or a hybrid cloud. This is an area that is quite well researched, it has been the centre of much high-quality research as is the case of the paper written by Yashpalsinh Jadeja and Kirit Modi that differentiates between the different approaches and benefits [11].

1) Public:

- **Definition:** Makes use of a public IaaS such as AWS web services.
- **Pros:** Managed via web browser interfaces or API's, fine-grained cost management pay for only what you use which helps reduce operations costs, elasticity properties that allow you to scale up and down depending on your needs.
- **Cons:** Security and privacy; often companies aren't comfortable handing over confidential data to another company mainly in the enterprise world. Often the IaaS providers reputation plays a big role when companies decide to take this approach. Replication time and cost; keep in mind that downtimes are still possible, public IaaS Servers can encounter the same problems as a company's internal servers.

2) Private:

- **Definition:** Implement using internal data centres, owned and managed by the company.
- **Pros:** Allow users to have higher control over their security and privacy, security is enhanced because only the company can access their private data, being one of the main reasons why the enterprise has been reticent on adhering to the public cloud and often decided to implement their own in-house solutions.
- **Cons:** Often require a huge amount of resources and management efforts, not as cost efficient during periods of lower usage.

3) Hybrid:

- **Definition:** Combination of public cloud and private cloud.
- **Pros:** Combines the benefits of both allowing companies to have more control over their privacy while still reducing resource wastage being able to make use of public services during times of higher demand.
- **Cons:** This is the most complex and hard to setup as it requires most of the work one would have to do for each of the other two approaches.

If an application does not require services provided by a specific IaaS provider and there aren't any business requirements that restrict to a particular approach, then companies may use any of the existing public IaaS solutions or a hybrid solution that makes use of different public IaaS and internal infrastructure. Much of the configuration work and services used vary across cloud providers, making it more difficult to change provider in the future. In order to use multiple cloud providers and prevent lock-in, companies must themselves build in support for multiple cloud providers, or find a solution that does for them.

In order to implement one of these approaches, one can use tools such as Terraform[12] for configuring, managing and automating infrastructure deployments. It requires serious consideration and extended infrastructure setup knowledge, for making architectural choices on matters such as High Availability, Failure recovery, Scalability and Networking. Taking Networking as an example traditionally, containers were exposed on the network via the shared host machine's address [13], but in a production environment, public traffic should not have access to certain containers, which should only be accessible by other containers within that private network. This had to be achieved purely using the IaaS virtual network support. Nowadays container orchestrators allow you to create private networks for each container. Still, the underlying infrastructure requires much thought to ensure the platform stays secure and is able to recover from unforeseen circumstances.

V. Cluster setup and Orchestration

Assuming a cloud implementation solution described in the previous section has been decided upon, the next step is to tackle the challenges associated with orchestration and cluster management. A container cluster architecture (see Fig 1) groups several nodes/host (Virtual instance and Bare-metal Server); each node is capable of running several services (Containerized applications) in simultaneous. The container orchestrators such as Swarm Mode by Docker [14] or Kubernetes by Google [15] is in charge of managing things such as:

- The volumes used by applications to store data that will persist, even after a container is terminated.
- Application deployment, scheduling and scaling across nodes.
- Service discovery and container networks, allowing for the logical network partition of containers by subnetworks and allowing containers communicate with each other.

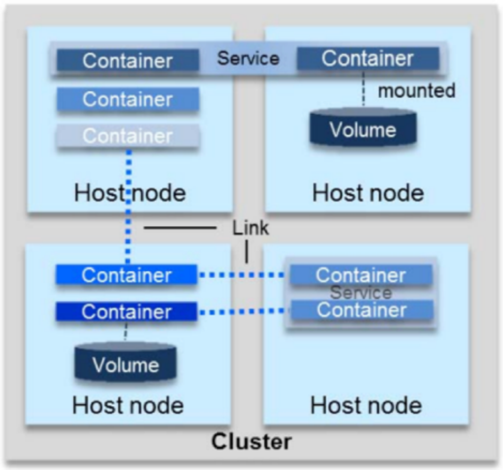

Fig. 1. General cluster architecture, from: Containers and Clusters for Edge Cloud Architecture [13].

Kubernetes has the steeper learning curve, installation and set-up can be complex, often requires the expertise of experienced platform admins. Swarm has the easier learning curve, it's built into Docker and integrates with the native Docker tools that developers also use in their local development environments, making it more desirable for smaller companies. Kubernetes tends to be the preferred solution because of its set of Deployment tools for facing many of the challenges of setting up and managing a production cluster [16], such as node autoscaling using Kubernetes Autoscale, a component that automatically adjusts the size of a Kubernetes Cluster, compatible with GCP, AWS and Azure. While with Swarm mode in order to archive autos calling you would have to implement your own solution for scaling up and down your cluster.

Much of the academic work on Orchestration such as the one referenced above "Containers and Clusters for Edge Cloud Architecture" [13] cover topics at a very high level, mentioning many of the Orchestration solutions available but fail to distinguish between different types of nodes, and to fully show the requirements, challenges and pitfalls associated with running them in production, for example, Swarm Mode differentiates between two types of nodes (see Fig 2) Managers and Workers [17], (while Kubernetes uses the notion of Master/Node). In swarm, a Single manager is labelled the leader and conducts orchestration tasks. Manager nodes manage the global cluster state using the Raft Consensus Algorithm, which is a protocol for implementing distributed consensus for leader election, log replication, and safety [18]. Worker nodes receive and execute tasks dispatched from manager nodes.

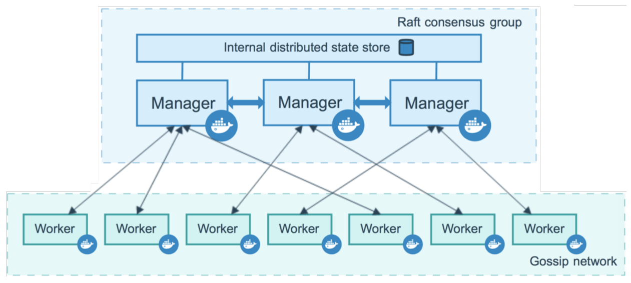

Fig. 2. Manager/Worker architecture, from: Official Docker Documentation [17].

According to the Docker Swarm documentation in order to deploy your application, you must send the request to a manager node, and for production-grade deployments, you should take follow these rules of thumb [19]:

- When a manager node fails, the number of failures tolerated by your swarm decreases. Don't leave that node offline for too long.
- You should distribute your manager nodes across different availability zones. This way your cluster can continue working even if an entire availability zone goes down.
- Adding many manager nodes to the cluster might lead to performance degradation, as changes to configurations need to be replicated across all manager nodes. The maximum advisable is seven manager nodes.

These rules have a major impact on the cluster architecture and the architecture of the underlying infrastructure. For example in order to use a continuous integration tool, one must ensure that it can talk to one of the swarm managers, while still ensuring it stays secure from the outside world.

VI. Volumes and Database Backups

Data stored in volumes may persist after a container is terminated, but data storage and volume management is still a problem in distributed clusters. A container can be deployed on a number of nodes in the cluster; often the volume may not be available on that node. There are volume managing plugins available that one could use to help manage volumes; another solution is the user of a data container (container with a mounted volume), that is directly linked and often follows the container running the application to the physical machine regardless of location in the cluster [13]. Data storage applications, such as databases can also be deployed in a container, and when set up properly they can be easily scaled to handle increased I/O based on demand, along with high availability.

Nodes could malfunction and die causing data to be lost forever. So a reliable backup/restore strategy is key to running modern containerized applications in production. The strategy selected for managing volumes and backups on production environments will positively or negatively impact the PaaS customers.

Official Ubuntu suggestions for managing backups [20]:

- Store backups somewhere separate from your computer on an external hard disk, for example.
- For maximum security, do not keep the backup in the same building as your computer. If they are kept together in the case of a fire or theft, both copies would be lost.
- Store backups on a device that has sufficient disk capacity, taking into account it will grow over time.
- Choose an appropriate backup medium, regardless of it being local or a remote storage option:
  - USB memory key (low capacity)
  - Internal disk drive (high capacity)
  - External hard disk (typically high capacity)
  - Network-connected drive (high capacity)
  - File/backup server (high capacity)
  - Writable CDs or DVDs (low/medium capacity)
  - Online backup service (Such as Amazon's S3, unlimited capacity with the only restriction being the cost).

VII. CI/CD and Image Management

Paul M. Duvall, Steve Matyas and Andrew Glover's definition of a simplified CI pipeline for software consists of 4 steps [21]:

- Developer commits code to the version control repository. While the CI server on continuously poll this repository for changes.
- When the CI server detects new code commits, it retrieves the latest copy of the code from the repository and then executes a build script, which integrates the software.
- The CI server generates some feedback, logs presented by some GUI or via email; the data usually consists of some form of test results.
- CI server goes back to continuously polling for changes in the code repository.

Typically there would also be a post-stage for Continuous Delivery (CD), that deploys the application onto a server. A delivery pipeline for containers is somewhat similar; it still makes use of a central version management code repository and a CI/CD server such as Jenkins or TravisCI. With the addition of some extra components, for storing images and scanning for security vulnerabilities. The DockerHub is great for distributing open source tooling but unless you plan to release your Docker image to the world, you'll also need to set up a private Docker image repository and to select an option to secure the private Docker images, while still making them accessible to the build and deployment processes.

VIII. Logging and Monitoring

Logging and monitoring are fundamental to the successful implementation of a PaaS solution; the amount of tooling involved in setting up your own PaaS tends to be so large that it becomes impossible to manually monitor all of them. Every company has different business requirements or preferences for different tools and since containers started gaining traction, even more, have come out. See Fig 3 taken from "Docker

Cluster Management for the Cloud", to get an idea of how big a problem this really is [22].

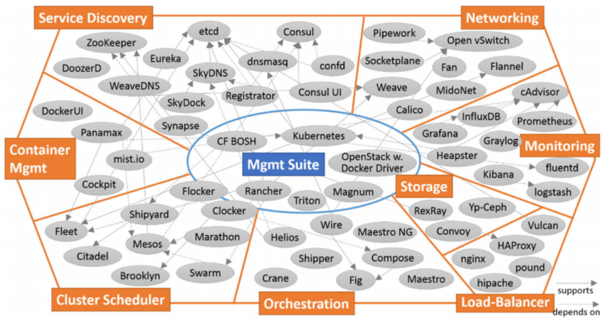

Fig. 3. Partial view of the Docker ecosystem [22].

In fact, all the services represented above are but a fraction of the number of services that form the Docker ecosystem. The lack of standards means that running and connecting many of these services varies from company to company, often involving large amounts of scripting. Logging and monitoring are the main point of access when things start to misbehave. Typically, logging and monitoring solutions are split into tools that collect and store data (for example, application logs and other resource usage data) and web-based front-ends for visualizing them. Some solutions can be used but in many cases, companies prefer to keep them separate [22].

A. *Logging*

Using Docker's logging features in development enables rapid troubleshooting, but when dealing with multiple container instances across a number of hosts, it becomes more difficult to track down issues. Properties of a logging system:

- Log collection
- Centralized aggregation
- Long-term retention/storage
- Log analysis
- Log searching and reporting.

Logging isn't just useful for troubleshooting purposes, when set-up properly it offers enormous business value; a good log management solution helps companies making business decisions and improving their products and services based on information such as the geo-identity of their user traffic data. There are a lot of log management solutions but many are costly or don't scale very well. A popular open source highly scalable solution is ELK [23]. The main components of ELK are Elasticsearch, Logstash and Kibana; together they form a very effective tool for interactively analysing log files.

Elasticsearch is a NoSQL database used for indexing, storage and retrieval. Elasticsearch exposes it's capabilities via a REST API using JSON, stores content using indexes and it looks to the indexes as opposed to the contents. Thus, making it very quick and scalable. Logstash is a tool for managing events and logs. It is used to collect and parse logs into JSON based on regexes before sending them to Elasticsearch. These logs can then be viewed and analysed using Kibana. Kibana is used for visualizing data stored in Elasticsearch using dashboards. Kibana dashboards are highly customizable allowing users to tweak them to their particular needs.

B. *Monitoring*

Monitoring is an essential part of cloud computing, even more so for container-based PaaS, where services frequently move between nodes. Monitoring should provide information about resource usage for the various system components and the ability to send alerts based on predefined thresholds. In some cases, monitoring may even be used as a basis for pay-per-use [22].

There is a large abundance of monitoring tools; some common solutions like Nagios, other more specialized within the Docker ecosystem, as is the case of Google's cAdvisor, designed for the specific purpose of monitoring containers. cAdvisor does not store historical data, hence it's Web UI is limited to the most recent data and tends to be used in conjunction. Grafana (web-based front-end similar to Kibana), Prometheus (advanced analytics and alerting) and InfluxDB or other time series databases as data stores [22].

Implementing an adequate monitoring strategy will help minimize the impact of outages and maximize resources, resulting in happier customers.

IX. Standardization

Containers are largely adopted for PaaS clouds [13]. In recent years technologies such as Docker or Kubernetes, slowly became de facto standards. There have also been some efforts from the IT industry, as is the case of the Open Container Initiative (OCI) [24]. The OCI is the governance structure that creates open industry standards for container formats and runtime. Established in 2015 by a group of companies seen as leaders in the container industry, the OCI currently contains two specifications:

- **image-spec:** Image Specification. The Runtime Specification outlines how to build an OCI Image, which can then be verified, discovered, downloaded, and unpacked into an OCI Runtime filesystem bundle.
- **runtime-spec:** The Runtime Specification outlines how to run an OCI Runtime filesystem bundle, that was previously unpacked on disk.

At a high level, any container platform (for example Docker) that makes use of an OCI compliant runtime, would be able to download an OCI Image and unpack it into an OCI Runtime filesystem bundle that can then be run as a container [24]. Despite the efforts of some companies to create container standards, this is still a serious problem that is making it difficult for smaller companies to adopt container technologies. The problem is that the number of services involved in building a PaaS solution is very large, the processes and tools used to set it all up vary on a company basis.

X. Conclusion

Container technology is a relatively new area with a huge potential to substantially advance PaaS technology. Much of the literature work available touches on the basic principle of container technology; although, very few actually dive into the challenges associated with the implementation of production ready container platforms.

Containers have a bright future especially in the PaaS use case, due to their inherent advantages over VMs in relation to performance improvements and reduced start-up time. However, some improvements are required in the areas of process standardization as well as an abstract development and architecture layer. There are many applications for dealing with different business requirements; setting these up as part of your PaaS solution usually involves a large amount of scripting, which is difficult to test. The lack of a standard way to set things up is making it difficult for smaller companies to adopt container technologies.

REFERENCES

[1] W. Vogels. (January/February 2008). *Beyond Server Consolidation*, Queue Virtualization Volume 6 Issue 1. ACM New York, NY, USA.

[2] Intel IT Center. (August 2013). *Virtualization and cloud computing*, Intel.

[3] R. Smith. (2017) *Docker Orchestration a concise, fast-paced guide to orchestrating and deploying scalable services with Docker*, Packt Publishing.

[4] Zhiyong Shan, Xin Wang, Tzi-cker Chiueh and Xiaofeng Meng. (2017). *Safe Side Effects Commitment for OS-Level Virtualization*, ICAC '11 Proceedings of the 8th ACM international conference on Autonomic computing. ACM New York, NY, USA.

[5] Docker. [Online]. Available: https://www.docker.com/ (accessed January 29, 2018)

[6] C. Boettiger. (January 2015). *An introduction to Docker for reproducible research.*, ACM SIGOPS Operating Systems Review. ACM New York, NY, USA.

[7] C. Pahl. (July 2015). *Containerization and the PaaS Cloud*, IEEE Cloud Computing Volume: 2, Issue: 3. IEEE.

[8] D. Beimborn, T. Miletzki and S. Wenze. (2011). *Platform as a Service (PaaS)*, Business & Information Systems Engineering: Vol. 3. Available at: http://aisel.aisnet.org/bise/vol3/iss6/6

[9] P. Clarke, R. O'Connor and P. Elger. (January 2017). *Technology Enabled Continuous Software Development*, 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery, IEEE.

[10] A. Tosatto, P. Ruiu and A. Attanasio. (August 2015). *Container-based orchestration in cloud: state of the art and challenges*, 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, IEEE.

[11] Y. Jadeja and K. Modi. (August 2015). *Container-based orchestration in cloud: state of the art and challenges*, 2012 International Conference on Computing, Electronics and Electrical Technologies, IEEE.

[12] HashiCorp. *Terraform*. [Online]. Available: https://www.terraform.io/ (accessed February 1, 2018)

[13] C. Pahl and B. Lee (August 2015). *Containers and Clusters for Edge Cloud Architectures – A Technology Review*, 2015 3rd International Conference on Future Internet of Things and Cloud, IEEE.

[14] Docker. *Swarm*. [Online]. Available: https://docs.docker.com/engine/swarm/ (accessed February 2, 2018)

[15] Kubernetes. [Online]. Available: https://kubernetes.io/ (accessed February 2, 2018)

[16] Platform9. *Kubernetes vs Docker Swarm* [Online]. Available: https://platform9.com/blog/kubernetes-docker-swarm-compared/ (accessed February 2, 2018)

[17] Docker. *How Swarm mode works*. [Online]. Available: https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/. (accessed February 3, 2018)

[18] D. Ongaro and J. Ousterhout. (August 2015). *In Search of an Understandable Consensus Algorithm*, Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference, USENIX.

[19] Docker. *Set up high availability*. [Online]. Available: https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/set-up-high-availability/#size-your-deployment (accessed February 4, 2018)

[20] Ubuntu. *Where to store your backup*. [Online]. Available: https://help.ubuntu.com/stable/ubuntu-help/backup-where.html (accessed February 4, 2018)

[21] P. M. Duvall, S. Matyas, and A. Glover (2007). *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series)*, ISBN: 0321336380. Addison-Wesley Professional.

[22] R. Peinl, F. Holzschuher and F. Pfitzer. (2016). *Docker Cluster Management for the Cloud - Survey Results and Own Solution*, Journal Grid Computing Volume 14. Springer-Verlag New York, Inc.

[23] T. Prakash, M. Kakkar and K. Patel. (2016). *Geo-Identification of Web Users through Logs using ELK Stack*, 2016 6th International Conference - Cloud System and Big Data Engineering. IEEE.

[24] Open Containers Initiative (OCI). [Online]. Available: https://www.opencontainers.org/ (accessed February 6, 2018)