2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments

Migrating from Virtualization to Dockerization in the Cloud: Simulation and Evaluation of Distributed Systems

Nitin Naik

Defence School of Communications and Information Systems
Ministry of Defence, United Kingdom
Email: nitin.naik100@mod.uk

***Abstract*—Virtualization is the nucleus of the cloud computing for providing its services on-demand. Cloud-based distributed systems are predominantly developed using virtualization technology. However, the requirement of significant resources and issues of interoperability and deployment make it less adoptable in the development of many types of distributed systems. Dockerization or Docker Container-based virtualization has been introduced in the last three years and gaining popularity in the software development community. Docker has recently introduced its distributed system development tool called Swarm, which extends the Docker Container-based system development process on multiple hosts in multiple clouds. Docker Swarm-based containerized distributed system is a brand new approach and needs to be compared with the virtualized distributed system. Therefore, this paper presents the simulation and evaluation of the development of a distributed system using virtualization and dockerization. This simulation is based on Docker Swarm, VirtualBox, Ubuntu, Mac OS X, *nginx* and *redis*. To simulate and evaluate the distributed system in the same environment, all Swarm Nodes and Virtual Machines are created using VirtualBox on the same Mac OS X host. For making this evaluation rational, almost similar system resources are allocated to both at the beginning. Subsequently, similar servers *nginx* and *redis* are installed on the Swarm Node and Virtual Machine. Finally, based on the experimental simulation results, it evaluates their required resources and operational overheads; thus, their performance and effectiveness for designing distributed systems.**

Keywords*—*Virtualization, Dockerization, Distributed Systems, Cloud, Virtual Machine Monitor, Hypervisor, Docker, Virtual Machine, Container

I. Introduction

Virtualization is the most efficacious technology to simulate a distributed environment for the development of distributed systems similar to the real production site [1]. It embeds a pre-production site into each developer's machine and offers them a real runtime environment for the system development [2], [3]. Thus, virtualization made system development more flexible and effective by allowing the dynamic allocation of hardware and software resources to support the build, integration, and test phases of complex system development projects [4], [5], [6]. However, the creation of virtualized environment requires lengthy and tedious installation and configuration process and highly skilled IT staff [7]. It also demands the good understanding of a wide range of technologies such as operating systems, databases, application servers and network services. This makes integration of virtualization technologies into the productive system development processes much harder [1]. Despite managing everything successfully, the biggest issues with the virtualization is its significant resource and operational overheads, which make it less adoptable in many types of distributed systems [8].

Dockerization or Docker Container-based virtualization has recently emerged as an alternate lightweight technology for the system development process and gaining popularity in the software development community [9], [10]. Docker offers the ability to package applications and their dependencies into lightweight Containers that move easily between different distros, start up quickly and are isolated from each other [10], [11]. It aims to address the challenges of resource, speed and performance of virtualization in the system development process [12], [13]. In addition to all these facilities, its greatest advantage is that it provides developer's workflow [14]. Thus, Docker Containers-as-a-Service (CaaS) platform empowers developers and sysadmins to build, ship and run distributed applications anywhere [15]. Docker recently launched its distributed system development tool called Swarm, which extends the Docker Container-based system development process on multiple hosts in multiple clouds.

Virtualization is a well-established technology in the development of cloud-based distributed systems [1], [2], [3], [16], [17]. Whereas dockerization is an effective system development tool and now gaining popularity in the software development community [10], [11], [12], [13], [14]. Docker Swarm-based containerized distributed system is a brand new approach and needs to be compared with virtualized distributed system. The selection of one of the technology for building distributed system is a difficult task. Therefore, this paper presents the simulation and evaluation of the development of a distributed system using virtualization and dockerization. This simulation is based on Docker Swarm, VirtualBox, Ubuntu, Mac OS X, *nginx* and *redis*. To simulate and evaluate the distributed system in the same environment, all Swarm Nodes and Virtual Machines are created using VirtualBox on the same Mac OS X host rather than on multiple hosts in multiple clouds. For making this evaluation rational, almost similar system resources are allocated to both at the beginning. Subsequently, similar servers *nginx* and *redis* are installed on the Swarm Node and Virtual Machine. Finally, based on the experimental simulation results, it evaluates their required

2326-6937/16 $31.00 © 2016 Crown Copyright
DOI 10.1109/MESOCA.2016.9

resources and operational overheads; thus, their performance and effectiveness for designing distributed systems.

The remainder of this paper is organised as follows: Section II explains the theoretical background of Virtualization and Dockerization techniques; Section III exhibits the actual implementation of Dockerization on a non-Linux machine (e.g., Mac OS X machine in this implementation) and how it differs from the Linux-based implementation; Section IV presents a simulation of the distributed system using Docker Swarm Nodes and Virtual Machines; Section V evaluates Virtualization and Dockerization techniques for the development of distributed systems based on the experimental simulation results. Section VI concludes the paper and suggests some future areas of extension.

II. Virtualization and Dockerization

This section presents the theoretical background of virtualization and dockerization.

A. Virtualization

Virtualization technique allows a user to create a virtual version of a device or resource, such as a server, storage device, network or operating system. It is an abstraction of the computer hardware that facilitates a single machine to act as if it where many machines. Virtualization framework divides the resource into one or more execution environments [18]. It enables multiple operating systems to run on the same physical platform. Virtualization is a combination of software and hardware engineering that creates Virtual Machines (VMs) [19]. Generally, virtualization architecture has two main components: Virtual Machine Monitor (VMM)/Hypervisor and Virtual Machine (VM) as shown in Fig. 1. The VMM/Hypervisor is normally the software which runs the Virtual Machine (VM)/Guest Computer. The VMM is the control system within the virtualization technology [19], and translation system between VMs and hardware infrastructure. VMM/Hypervisor can be classified into two types: Type 1 or bare metal hypervisor and Type 2 hypervisor. Type 1 runs directly on the hardware of the host without the need of a host operating system. Whereas, Type 2 runs on top of a host operating system and then spawns higher level Virtual Machines. A Virtual Machine (VM) is a self-contained operating environment that behaves as if it is a separate computer. Generally, every Virtual Machine has its own operation system, binaries, libraries and applications as shown in Fig. 1. Virtualization technique requires substantial bandwidth, storage and processing capacities. A large server or desktop is needed, if it is going to host multiple running Virtual Machines [20]. There are so many popular virtualization technologies available such as VMWare, VirtualBox, Parallels, QEMU, UML and Xen. In the rest of the paper, virtualization and Virtual Machine terminology represent the Type 2 hypervisor category because this architecture is comparable to dockerization architecture.

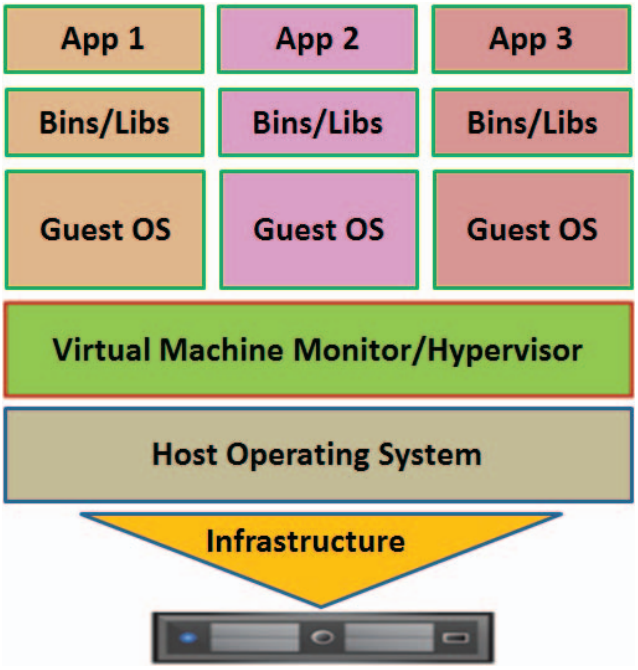

Fig. 1. Virtualization and Virtual Machine

B. Dockerization

Containers provide an isolated environment akin to virtualization but without virtual hardware emulation [9]. The concept of a Container is quite old in computing, however, they were never employed at large-scale. Containers run in user space on top of an operating system's kernel; therefore, Container virtualization is known as OS-level virtualization [21]. Container technology allows multiple isolated user space instances to be run on a single host [21]. Containers can also be classified into two categories System Containers and Application Containers. A System Container is similar to a full OS and runs all process such as init, inetd, sshd, syslogd, and cron. Whereas, Application Container only runs an application. Both types are useful in different circumstances [22]. There are many popular Container technologies available such as OpenVZ, LXC, and Solaris Zones, systemd-nspawn, lmctfy, Warden [21], [22]. Docker technology is an example of Container virtualization, and its process is known dockerization.

Docker is an open-source engine that automates the deployment of applications into Containers [21]. It is developed by Docker, Inc., which was previously known as dotCloud, Inc. that was one of the pioneering company in Platform-as-a-service market. Docker provides an isolated Container (see Fig. 2) based on a major Linux kernel feature known as *cgroups* and *namespace*. Consequently, each Container can have strong isolation, own network stack, storage stack, file system and resource management capabilities to allow friendly co-existence of multiple Containers on a single host [21]. A Container does not have its own operating system as it shares the same kernel; however, it contains all binaries and libraries to run an application inside it as shown in Fig. 2.

Docker Swarm: Distributed applications need distributed system and compute resources on it. Docker Swarm is a clustering and scheduling tool, which offers functionalities to turn a group of Docker Nodes into a virtual Docker System [23]. It builds a cooperative group of systems that can provide redundancy if one or more nodes fail. Swarm provides

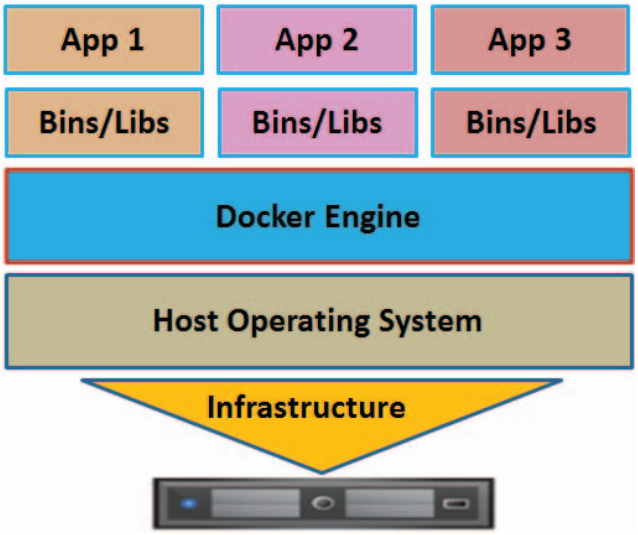

Fig. 2. Dockerization and Docker Container

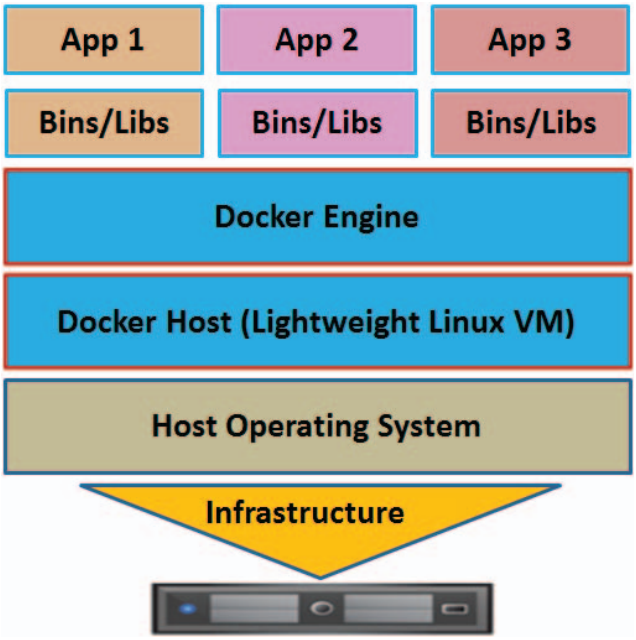

Fig. 3. Dockerization and Docker Container for Non-Linux Machine (Mac OS X Machine)

workload balancing for Containers. It assigns Containers to underlying nodes and optimizes resources by automatically scheduling Container workloads to run on the most appropriate host with adequate resources while maintaining necessary performance levels [24]. An IT administrator or developer controls Swarm using a swarm manager, which organises and schedules Containers. The swarm manager allows them to create a primary manager instance and multiple replica instances in case the primary instance fails [24].

III. Dockerization on Non-Linux Machine (Mac OS X Machine)

The previous section has explained the Docker architecture, but that was mostly for Linux-based machines, where the host Linux OS also plays a part of Docker Host. This section reveals the actual implementation of Docker Containers on a non-Linux machine (e.g., Mac OS X machine) and how it differs from the Linux-based implementation. It is important to know about this implementation because this paper examines Docker on Mac OS X machine. In non-Linux OS-based systems, Docker needs an additional component called Docker Host as shown in Fig. 3. Essentially, Docker Host is a lightweight Virtual Machine, which requires very few resources and operational overheads. In a Mac OS X installation, the Docker Engine runs inside a Linux VM called "default". The "default" is a lightweight Linux Virtual Machine made specifically to run the Docker engine on a non-Linux machine (e.g., Mac OS X machine). The great success of Docker is this small Virtual Machine, which runs completely in RAM and loads only in few seconds. This is largely due to its very small size (i.e., 35 MB in this implementation). Finally, the detailed architecture is shown in Fig. 3, which will be utilised in this experimental simulation.

IV. Experimental Simulation: Development of a Distributed System Using Virtualization and Dockerization

This simulation is based on Docker Swarm, VirtualBox, Ubuntu, Mac OS X, *nginx* and *redis*. To simulate and evaluate the distributed system in the same environment, all Swarm Nodes and Virtual Machines are created using VirtualBox on the same Mac OS X host rather than on multiple hosts in multiple clouds.

A. Experimental Simulation: Development of a Distributed System using Dockerization

This simulation demonstrates the development of a distributed system using dockerization. In this simulation, a sample distributed system is developed using Docker Swarm, *nginx* and *redis*. This distributed system is implemented as a cluster of four Swarm Nodes (Virtual Machines) on the same host computer to measure the performance in the same environment. However, this simulation can be implemented and extended to develop the distributed system on Docker supported multiple clouds such as Amazon Web Services, Microsoft Azure, Digital Ocean, Exoscale, Google Compute Engine [25]. The complete system development process is carried out on Mac OS X, therefore, it uses the non-Linux architecture of dockerization technology. Fig. 4 shows the experimental architecture of the distributed system using Docker Swarm Nodes.

Initially, Docker is running in the lightweight Virtual Machine called "default" as shown in Fig. 5.

For creating a Swarm cluster of nodes, the first step is to create a Docker Swarm image as shown in Fig. 6, which can be used as a discovery token to connect all nodes within the

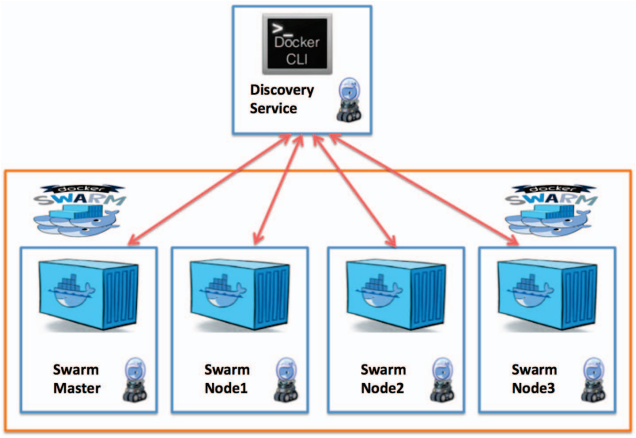

Fig. 4. Docker Swarm Architecture implemented in the simulation

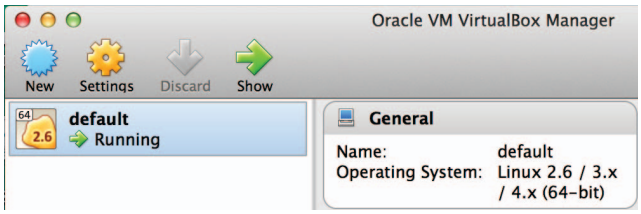

Fig. 5. Docker Engine running in "default" Virtual Machine

Swarm cluster. This simulation uses the Docker Hub based token discovery service.

Now use this Swarm image token to create all Swarm Nodes including Swarm Master in VirtualBox as shown in Fig. 7. In this simulation, Oracle VirtualBox driver is used, however, any other Docker supported driver can be used. Docker currently supports many virtual and cloud environments as a driver such as Amazon Web Services, Microsoft Azure, Digital Ocean, Exoscale, Google Compute Engine, Generic, Microsoft Hyper-V, OpenStack, Rackspace, IBM Softlayer, VMware vCloud Air, VMware Fusion, VMware vSphere [25]. In this distributed system, one Swarm master and three Swarm nodes are created as shown in Fig. 7. These Swarm nodes can be managed on different clouds by just changing the driver name to the desired cloud name from the above list in Fig. 7.

Once all Swarm nodes are created and running, they can be seen in VirtualBox Manager as shown in Fig. 8.

For this Swarm cluster up and running, next step is to set the environment variable as shown in Fig. 9. Now all the Swarm nodes and their details can be checked using the command shown in Fig. 9.

```
Nitins-MacBook-Pro:~ nitinnaik$ docker run swarm create
Unable to find image 'swarm:latest' locally
latest: Pulling from library/swarm
51436fd4bb0d: Pull complete
c31a5390266f: Pull complete
e40019be13ea: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:3add485cb6bb71c7113243753c5f484561549d9f782154b1c809219c9754ce46
Status: Downloaded newer image for swarm:latest
1e819d13e35941894c20e4f7c8d7bcb2
```

Fig. 6. Creating Docker Swarm image for using as a service discovery token

```
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver=virtualbox --swarm --swarm-master
--swarm-discovery=token://1e819d13e35941894c20e4f7c8d7bcb2 my-swarm-master
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver=virtualbox --swarm
--swarm-discovery=token://1e819d13e35941894c20e4f7c8d7bcb2 my-swarm-node1
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver=virtualbox --swarm
--swarm-discovery=token://1e819d13e35941894c20e4f7c8d7bcb2 my-swarm-node2
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine create --driver=virtualbox --swarm
--swarm-discovery=token://1e819d13e35941894c20e4f7c8d7bcb2 my-swarm-node3
```

Fig. 7. Creating Docker Swarm Master and Worker Nodes

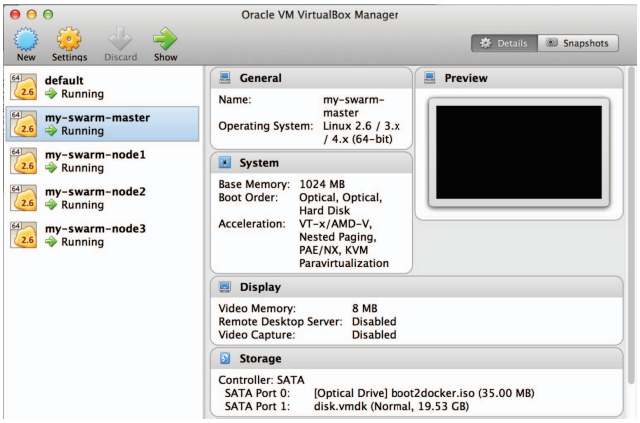

Fig. 8. Docker Swarm Master and Worker Nodes in VirtualBox Manager

Making this Swarm cluster as a working distributed system, it requires certain servers to be installed such as *nginx* and *redis* in this implementation. Here, two servers *nginx* and *redis* are started as two separate Containers on each Swarm nodes. This typical configuration is just to demonstrate the scheduling and scaling features of Swarm, however, in a real system, only one server can be used. Firstly,*nginx* servers are started on all the Swarm nodes as shown in Fig. 10. All these *nginx* Containers are automatically allocated by Swarm-agent to different Swarm nodes.

Finally, *redis* servers are started on all the Swarm nodes as shown in Fig. 11. Again, all these *redis* Containers are automatically allocated by Swarm-agent to different Swarm nodes. Now the list of all the running Containers with *nginx* and *redis* servers on different Swarm nodes is given in Fig. 12. This Docker Swarm-based distributed system can be used

```
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine env --swarm my-swarm-master
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.101:3376"
export DOCKER_CERT_PATH="/Users/nitinnaik/.docker/machine/machines/my-swarm-master"
export DOCKER_MACHINE_NAME="my-swarm-master"
# Run this command to configure your shell:
# eval $(docker-machine env --swarm my-swarm-master)
Nitins-MacBook-Pro:~ nitinnaik$ eval $(docker-machine env --swarm my-swarm-master)
Nitins-MacBook-Pro:~ nitinnaik$ docker-machine ls
NAME             ACTIVE      DRIVER       STATE     URL                         SWARM                       DOCKER
default          -           virtualbox   Running   tcp://192.168.99.100:2376                               v1.11.1
my-swarm-master  * (swarm)   virtualbox   Running   tcp://192.168.99.101:2376   my-swarm-master (master)    v1.11.1
my-swarm-node1   -           virtualbox   Running   tcp://192.168.99.102:2376   my-swarm-master             v1.11.1
my-swarm-node2   -           virtualbox   Running   tcp://192.168.99.103:2376   my-swarm-master             v1.11.1
my-swarm-node3   -           virtualbox   Running   tcp://192.168.99.104:2376   my-swarm-master             v1.11.1
```

Fig. 9. Setting environment for Swarm and listing details of all the Nodes

```
Nitins-MacBook-Pro:~ nitinnaik$ docker run -itd --name my-web1  nginx
56823827bcecfe0effd58dc3c7cffd1d067c1e0e0c192e217d6fa453af9eca23
Nitins-MacBook-Pro:~ nitinnaik$ docker run -itd --name my-web2  nginx
9ff0aec316eae172ae51f75003200b0f21332dc6d0fc2d2dfc5b542c2ab3d560
Nitins-MacBook-Pro:~ nitinnaik$ docker run -itd --name my-web3  nginx
5d6afec838e274443b84b4384e76e92947c2be278d0b4d22e1cdacddb9bca303
Nitins-MacBook-Pro:~ nitinnaik$ docker run -itd --name my-web4  nginx
40c958345f1c1b0dccb1d44efd953a56c6131cd886aac82077703cbcb9925c11
Nitins-MacBook-Pro:~ nitinnaik$ docker ps
CONTAINER ID  IMAGE  COMMAND                  CREATED              STATUS              PORTS               NAMES
40c958345f1c  nginx  "nginx -g 'daemon off"   About a minute ago   Up About a minute   80/tcp, 443/tcp     my-swarm-node3/my-web4
5d6afec838e2  nginx  "nginx -g 'daemon off"   10 minutes ago       Up 10 minutes       80/tcp, 443/tcp     my-swarm-node1/my-web3
9ff0aec316ea  nginx  "nginx -g 'daemon off"   12 minutes ago       Up 12 minutes       80/tcp, 443/tcp     my-swarm-master/my-web2
56823827bcec  nginx  "nginx -g 'daemon off"   14 minutes ago       Up 14 minutes       80/tcp, 443/tcp     my-swarm-node2/my-web1
```

Fig. 10. Running Containers with *nginx* server on all the Swarm Nodes

```
Nitins-MacBook-Pro:~ nitinnaik$ docker run -d --name my-web5  redis
b4cb7a57751e09753d4093f535ccb3136dcb1bfbe24d197bab6a4864b3f55e6a
Nitins-MacBook-Pro:~ nitinnaik$ docker run -d --name my-web6  redis
41a7d4d7f204602926530e256d7eed4534ea3472e11b582655a7e8ae66415e55
Nitins-MacBook-Pro:~ nitinnaik$ docker run -d --name my-web7  redis
c214ffb0fe3fb54e90774fb74a072e65a5a3b0bc9b747c54fe56763badb3109f
Nitins-MacBook-Pro:~ nitinnaik$ docker run -d --name my-web8  redis
d2c6e88e1e072275a1f1c0fc21e33c401c3da52f79563e08d2ac6efaa6cb5734
```

Fig. 11. Running Containers with *redis* server on all the Swarm Nodes

```
Nitins-MacBook-Pro:~ nitinnaik$ docker ps
CONTAINER ID   IMAGE   COMMAND                  CREATED          STATUS          PORTS               NAMES
d2c6e88e1e07   redis   "docker-entrypoint.sh"   8 seconds ago    Up 8 seconds    6379/tcp            my-swarm-master/my-web8
c214ffb0fe3f   redis   "docker-entrypoint.sh"   2 minutes ago    Up 2 minutes    6379/tcp            my-swarm-node2/my-web7
41a7d4d7f204   redis   "docker-entrypoint.sh"   3 minutes ago    Up 3 minutes    6379/tcp            my-swarm-node1/my-web6
b4cb7a57751e   redis   "docker-entrypoint.sh"   5 minutes ago    Up 5 minutes    6379/tcp            my-swarm-node3/my-web5
40c9583451c   nginx   "nginx -g 'daemon off"   9 minutes ago    Up 9 minutes    80/tcp, 443/tcp     my-swarm-node3/my-web4
5d6afec830e2   nginx   "nginx -g 'daemon off"   18 minutes ago   Up 18 minutes   80/tcp, 443/tcp     my-swarm-node1/my-web3
9ff0aec316ea   nginx   "nginx -g 'daemon off"   20 minutes ago   Up 20 minutes   80/tcp, 443/tcp     my-swarm-master/my-web2
56823827bcec   nginx   "nginx -g 'daemon off"   22 minutes ago   Up 22 minutes   80/tcp, 443/tcp     my-swarm-node2/my-web1
Nitins-MacBook-Pro:~ nitinnaik$
```

Fig. 12. All the running Containers with *redis* and *nginx* servers on all the Swarm Nodes

to develop distributed applications in the cloud environment.

B. Experimental Simulation: Development of a Distributed System using Virtualization

This simulation is similar to the dockerization-based simulation with an almost similar environment for making a rational comparison between the two technologies. Here, the similar distributed system is developed using VirtualBox, Ubuntu, *nginx* and *redis*. This distributed system is implemented as a cluster of four Virtual Machines on the same host computer to measure the performance in the same environment. However, this simulation can be implemented and extended to develop the distributed system on multiple clouds in the same way. The complete system development process is carried out on Mac OS X, and its experimental architecture using virtualization technology is shown in Fig. 13.

The creation process of a Virtual Machine in VirtualBox is relatively very easy and well known, therefore, it is omitted here. Once it is created, it can be seen in VirtualBox Manager as shown in Fig. 14. All the Virtual Machines are configured with Ubuntu Operating System (OS) to create the similar environment for both technologies.

Next step is to install *nginx* server on this Ubuntu-based Virtual Machine1 as shown in Fig. 15.

Subsequently, another *redis* server is installed on this Ubuntu-based Virtual Machine1 as shown in Fig. 16.

Similarly, all the Virtual Machines (2,3,4) can be easily created and required software can be installed. All the four Virtual

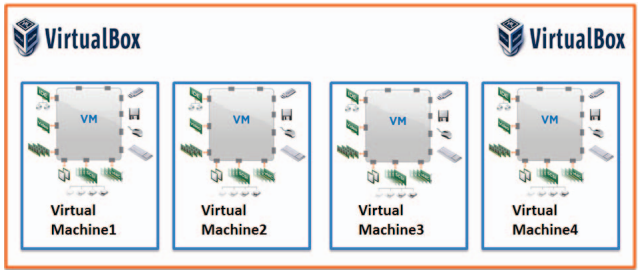

Fig. 13. Virtual Machine Architecture implemented in the simulation

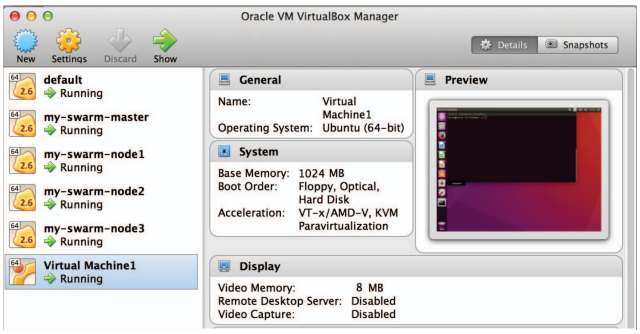

Fig. 14. Created Virtual Machine1 in VirtualBox Manager

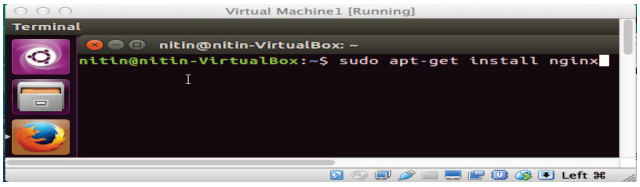

Fig. 15. Installing *nginx* server on Ubuntu-based Virtual Machine1

Machines (1,2,3,4) are shown in Fig. 17. Within this Virtual Machine-based distributed system, cooperation and application development would be similar to a stand alone machine-based activity and one Virtual Machine can be promoted to the master or domain controller to coordinate all other machines. This process is well known and thoroughly explained in literature [26], [27].

V. Experimental Results and Evaluation of Virtualization and Dockerization Technologies for the Development of a Distributed System

Designing distributed systems in the cloud using virtualization and dockerization is a wide topic and includes several QoS parameters such as resource and operational overheads, scalability, portability, interoperability, isolation, inter-communication and security [13], [22], [28]. This paper mainly focuses on the system performance and operational overheads of Docker Swarm Node-based and Virtual Machine-based distributed systems. Therefore, only limited simulation is carried out for obtaining relevant results related to required resources and operational overheads. These parameters are recorded during the each simulation process. Table I shows system configurations for Swarm Nodes and Virtual Machines, which exhibits that the equal system resources are allocated for building the similar size of Swarm Node and Virtual Machine for making this comparative evaluation rational. Table

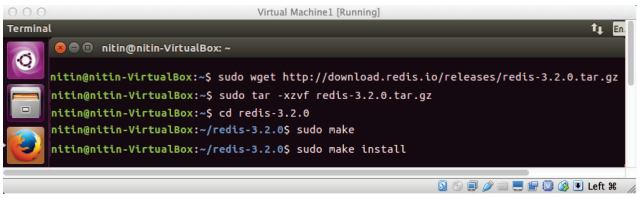

Fig. 16. Installing *redis* server on Ubuntu-based Virtual Machine1

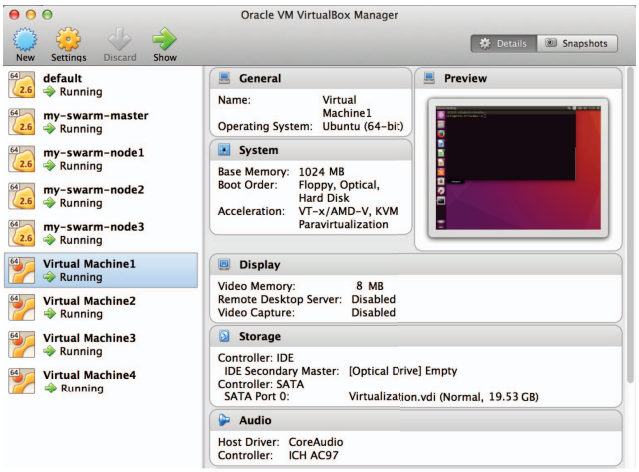

Fig. 17. Four Virtual Machines in VirtualBox Manager

II shows the average results obtained from both experimental simulations. Overall results in Table II illustrate that dockerization has consumed fewer resources and operational overheads as compared to virtualization. The dockerization-based distributed system utilised less memory, storage, CPU usage, CPU time, boot time and energy within the same Linux-based environment.

Memory is one of the most important metrics for the measurement of the performance of virtualization and dockerization. Docker Swarm Node has better memory usages as compared to Virtual Machine. The Docker Swarm Node is mostly used to run Application Container and a single process in it [22]. A standard Docker Container can only run a process CMD command and all processes that it spawns [29]. As an Application Container, unlike OS, it does not waste memory on redundant management processes [22]. However, an Application Container could not run several important system services automatically, and this gradually increases extra overheads to start these services and impediments in the system development at a later stage [29]. Alternatively, a full Linux OS runs all kinds of important system services by default, which are required for the effective system development process in any application area.

Storage is another crucial metric to determine the performance of virtualization and dockerization. A similar result is obtained for this metric, and the Docker Swarm Node is very compact requiring less storage. The main reason for higher storage overheads in the virtualization-based development process is due to the large size of the base image because it is a full OS image. Whereas, Docker Swarm Node needs a very small size of the base image as it comes with bare minimum features. This large image size in virtualization causes the problem of large system/application image which is difficult to implement or deploy [30]. This is one of the reasons of the popularity of the compact Docker Swarm Node among the software community. Indeed, this issue is not a big issue and can be easily resolved in virtualization. While maintaining the full functionality of Linux OS, the base image size can be reduced in virtualization by using relatively smaller Linux distros such as TinyCore, Lubuntu, LXLE, MX Linux, Porteus, Slitaz, Vector Linux, Absolute Linux and Puppy Linux depending on the requirement of the distributed system [31].

Another decisive performance metric is CPU usage and time; again the Docker Swarm Node has achieved better performance results over the Virtual Machine. Therefore, Docker Container is beneficial for CPU intensive application. However, Docker Container is a locked down environment with no direct access to many kernel resources; and it contains multiple abstraction layers. Consequently, large I/O and network usage may consume more resources because of Docker-related abstraction layers demanding significant context switching between kernel and userspace [32]. Similarly, depending on the nature of an application, virtualization can be beneficial or expensive. Initial boot time of virtualization can also be reduced automatically if the lightweight version of Linux is used in the distributed system.

While evaluating the other non-functional properties for both, Docker Swarm has inbuilt features of scheduling, load balancing, discovery, availability and networking that help it to act as a natural distributed system, which are demonstrated in the simulation. The token discovery services are shown in Fig. 6 [33]; scheduling, load balancing, and availability characteristics are shown in Figs. 10 to 12, where Docker Swarm managed eight Containers automatically. Whereas, Virtual Machines are completely independent from each other and, therefore, as a stand-alone machine, they require extra/external provision to coordinate and act as a distributed system similar to the Swarm-based distributed system. This would be an initial advantage of Docker Swarm over on Virtual Machine. However, when comparing security and privacy, Containers may be less secure than Virtual Machines as they share kernel resources and application libraries on the same virtual host, and the Virtual Machine has the advantage of having hardware isolation [34], [35], [36]. However, in the near future, Docker Container can be made more secure using the traditional OS security mechanism to lock down the environment inside a Container.

This particular evaluation outlines the requirement of fewer resources and overheads for the Docker Swarm-based distributed system as compared to Virtual Machine-based distributed system. However, it can not be taken as a general guidance for all kinds of system. The main limitation of this simulation is that it is a very small system and may not reflect the requirements of a wide range of systems. Therefore, the use of virtualization and dockerization can be determined precisely depending on the need of a specific system. Despite Docker Swarm has demonstrated better performance, its Containers and images can be run only under dockerized GNU/Linux, and an application has to be Docker locked-in. Additionally, it is supported by a handful number of cloud computing corporations. Finally, dockerization technology is mainly focused on PaaS (Platform-as-a-Service), and its prime objective is the deployment of distributed systems (software/applications) with portability and interoperability while utilising operating systems (OS) virtualization principles [37], [38], [26]. Virtualization technology is mainly focused on IaaS (Infrastructure-as-a-Service), and its prime objective is the deployment of distributed systems (hardware provisioning/allocation and management) while utilising hardware virtualization principles [27], [37].

TABLE I. SYSTEM CONFIGURATIONS FOR THE VIRTUAL MACHINE AND DOCKER SWARM NODE OF DISTRIBUTED SYSTEM

System Parameters	Virtual Machine		Docker Swarm Node	
	Single VM	**Distributed System**	**Single Node**	**Distributed System**
1. Configured Base Memory	1 GB x 4 =	4 GB	1 GB x 4 =	4 GB
2. Configured Storage Memory	19.53 GB x 4 =	78.12 GB	19.53 GB x 4 =	78.12 GB
3. Configured Video Memory	8 MB x 4 =	32 MB	8 MB x 4 =	32 MB
4. ISO Image Size	1.49 GB x 4 =	5.96 GB	35.0 MB x 4 =	140 MB
5. ISO Image Version	Ubuntu OS 16.04	Ubuntu OS 16.04	Docker 1.11.1 (Boot2Docker)	Docker 1.11.1 (Boot2Docker)

TABLE II. RESOURCES AND OPERATIONAL OVERHEADS FOR THE VIRTUAL MACHINE AND DOCKER SWARM NODE OF DISTRIBUTED SYSTEM

Performance Metrics	Virtual Machine		Docker Swarm Node	
	Single VM	**Distributed System**	**Single Node**	**Distributed System**
1. Real Memory Size	2.73 GB x 4 =	10.92 GB	1.04 GB x 4 =	4.16 GB
2. Virtual Memory Size	5.25 GB x 4 =	21 GB	3.52 GB x 4 =	14.08 GB
3. Shared Memory Size	46.2 MB x 4 =	184.8 MB	6.3 MB x 4 =	25.2 MB
4. Private Memory Size	2.60 GB x 4 =	10.4 GB	1.03 GB x 4 =	4.12 GB
5. %CPU	10.54% x 4 =	42.16%	2.14 % x 4 =	8.56%
6. Boot Time	16.14 s x 4 =	64.56 s	3.75 s x 4 =	15 s
7. Energy Impact	6.47 x 4 =	25.88	1.16 x 4 =	4.64
8. Machine/Node Image Size	1.73 GB x 4 =	6.92 GB	544.32 MB x 4 =	2.13 GB

VI. CONCLUSION

This paper presented the simulation and evaluation of the development of a distributed system using virtualization and dockerization. This simulation was based on Docker Swarm, VirtualBox, Ubuntu, Mac OS X, *nginx* and *redis*. To simulate and evaluate the distributed system in the same environment, all Swarm Nodes and Virtual Machines were created using VirtualBox on the same Mac OS X host. For making this evaluation rational, almost similar system resources are allocated to both at the beginning. Subsequently, similar OS environment, and servers *nginx* and *redis* are installed on Swarm Node and Virtual Machine. Finally, based on the experimental simulation results, it evaluated their required resources and operational overheads; thus, their performance and effectiveness for designing distributed systems.

The overall result showed that the Docker Swarm-based distributed system consumed fewer resources and operational overheads as compared to the Virtual Machine-based distributed system. Additionally, Docker Swarm has inbuilt features of scheduling, load balancing, discovery, availability and networking that help it to act as a natural distributed system. Whereas, Virtual Machines are completely independent from each other and, therefore, as a stand-alone machine, they require extra/external provision to coordinate and act as a distributed system similar to Docker Swarm. However, this result is based on the simulation of a very small distributed system and may not cover the requirements of a wide range of distributed systems. Therefore, the use of virtualization and dockerization can be determined precisely depending on the need of a specific type of system. In the future, it may be interesting to simulate and evaluate dockerization and virtualization technologies in the cloud, while using some most lightweight Linux operating systems on the Virtual Machine.

REFERENCES

[1] J. C. Dueñas, F. Cuadrado, B. García, H. A. Parada, and J. L. Ruiz, "System virtualization tools for software development," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 52–59, 2009.

[2] S. Seetharaman and K. Murthy, "Test optimization using software virtualization," *Software, IEEE*, vol. 23, no. 5, pp. 66–69, 2006.

[3] G.-H. Kim, Y.-G. Kim, and K.-Y. Chung, "Towards virtualized and automated software performance test architecture," *Multimedia Tools and Applications*, vol. 74, no. 20, pp. 8745–8759, 2015.

[4] L. M. Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak, "Using virtualization to improve software rejuvenation," in *Network Computing*

and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on. IEEE, 2007, pp. 33–44.

[5] N. Bonfiglio, A. Ryan, Y. Zhang, and D. Mercer, "Systems and methods for on-demand deployment of software build and test environments," Apr. 18 2007, US Patent App. 11/788,219.

[6] S. I. Larimore, C. M. Murphey, and K. C. Obata, "Method and system for virtualization of software applications," Dec. 8 2015, US Patent 9,207,934.

[7] A. Aguiar and F. Hessel, "Embedded systems' virtualization: The next challenge?" in *Rapid System prototyping (RSP), 2010 21st IEEE International symposium on*. IEEE, 2010, pp. 1–7.

[8] M. Janssen and A. Joha, "Challenges for adopting cloud-based software as a service (SAAS) in the public sector." in *ECIS*, 2011.

[9] N. Naik, "Building a virtual system of systems using Docker Swarm in multiple clouds," in *IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016.

[10] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[11] G. M. Tihfon, J. Kim, and K. J. Kim, "A new virtualized environment for application deployment based on docker and aws," in *Information Science and Applications (ICISA) 2016*. Springer, 2016, pp. 1339–1349.

[12] C. Anderson, "Docker [Software Engineering]," *IEEE Software*, no. 3, pp. 102–c3, 2015.

[13] P. Marinescu, P. Hosek, and C. Cadar, "Covrig: A framework for the analysis of code, test, and coverage evolution in real software," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 93–104.

[14] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D'Souza, S. Devoid, D. Murphy-Olson, N. Desai *et al.*, "Skyport: container-based execution environment management for multi-cloud scientific workflows," in *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*. IEEE Press, 2014, pp. 25–32.

[15] Docker.com. (2016) Docker use cases. [Online]. Available: https://www.docker.com/use-cases

[16] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[17] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[18] Csfi.com. (2016) Virtualization. [Online]. Available: http://www.csfi.com/index.php/virtualization/

[19] T. Burger. (2012, March 5) The advantages of using virtualization technology in the enterprise. [Online]. Available: https://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise

[20] B. Kirsch. (2014) Virtual machine. [Online]. Available: http://searchservervirtualization.techtarget.com/definition/virtual-machine

[21] J. Turnbull, *The Docker Book: Containerization is the new Virtualization*. James Turnbull, 2014.

[22] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015, pp. 171–172.

[23] Docker.com. (2016) Docker Swarm. [Online]. Available: https://www.docker.com/products/docker-swarm

[24] M. Rouse. (2016) Docker swarm. [Online]. Available: http://searchitoperations.techtarget.com/definition/Docker-Swarm

[25] Docker.com. (2016) Supported drivers. [Online]. Available: https://docs.docker.com/machine/drivers/

[26] D. I. Savchenko and G. I. Radchenko, "Mjolnirr: private PaaS as distributed computing evolution," in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*. IEEE, 2014, pp. 386–391.

[27] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[28] K.-T. Seo, H. Hwang, I. Moon, O. Kwon, and B. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud," *Advanced Science and Technology Letters*, vol. 66, pp. 105–111, 2014.

[29] Phusion.github.io. (2016) Your docker image might be broken without you knowing it. [Online]. Available: http://phusion.github.io/baseimage-docker/

[30] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.

[31] S. Sharma and N. Peers. (2016, January 28) 10 of the most popular lightweight linux distros. [Online]. Available: http://www.techradar.com/news/software/operating-systems/10-of-the-most-popular-lightweight-linux-distros-1295034

[32] Boycottdocker.org. (2016) Boycott Docker. [Online]. Available: http://www.boycottdocker.org/

[33] Docker.com. (2016) Docker Swarm. [Online]. Available: https://docs.docker.com/v1.5/swarm/

[34] N. Naik and P. Jenkins, "A secure mobile cloud identity: Criteria for effective identity and access management standards," in *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 2016, pp. 89–90.

[35] ——, "An analysis of open standard identity protocols in cloud computing security paradigm," in *14th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2016)*. IEEE, 2016.

[36] K. Collins. (2016) When to use containers or virtual machines, and why. [Online]. Available: http://www.nextplatform.com/2015/08/06/containers-versus-virtual-machines-when-to-use-each-one-and-why/

[37] C. Pahl, "Containerization and the PaaS cloud," *IEEE Cloud Computing*, no. 3, pp. 24–31, 2015.

[38] C. Ruiz, S. Harrache, M. Mercier, and O. Richard, "Reconstructable software appliances with kameleon," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 80–89, 2015.