Jurnal Informatika Terpadu

https://journal.nurulfikri.ac.id/index.php/JIT
ISSN ONLINE : 2460-8998

PENGUJIAN DAN IMPLEMENTASI EFEKTIVITAS SISTEM VIRTUALISASI BERBASIS DOCKER PLATFORM

Husni Mubarok[1], Henry Saptono[2]

[1, 2]Teknik Informatika, Sekolah Tinggi Teknologi Terpadu Nurul Fikri
Jakarta Selatan, DKI Jakarta, Indonesia 12640
husnimubarok295@gmail.com, henry@nurulfikri.co.id

Abstract

The increasing number of application development needs require various and different services with different environments, so Docker-based virtualization is needed so that applications can be run on various platforms so that it can make it easier for application developers and system administrators to build and run applications that have been implemented designed, namely by running the required services in a container. This scientific writing intends to analyze and design a Docker-based virtualization platform installed on the Ubuntu 17.10 Linux operating system, including the process of installing, configuring, and running the Docker application. Docker is an open project that works to help developers or system administrators build, package, and run applications anywhere in a container. The architecture uses the client-server method, the docker client, sends requests to the Docker daemon to build, distribute and run Docker containers. Both the Docker client and the Docker daemon communicate via a socket using the Restful API. Docker provides tools and platforms to manage the lifecycle of containers where developers and system administrators can develop applications and their supporting components using containers. A container is a container into a unit to distribute and test applications ready to be installed.

Keywords: Docker, *Client Server, Container, Docker Daemon, Socket, Restful* API

Abstrak

Semakin banyaknya kebutuhan pengembangan aplikasi yang membutuhkan servis yang beragam dan berbeda-beda dengan lingkungan yang berbeda juga, maka dibutuhkan adanya virtualisasi dengan berbasis Docker agar aplikasi dapat dijalankan di berbagai *platform* yang berbeda sehingga dapat memudahkan pengembang aplikasi dan sistem administrator dalam membangun dan menjalankan aplikasi yang telah dirancang, yaitu dengan menjalankan servis yang dibutuhkan dalam sebuah kontainer. Penulisan ilmiah ini bermaksud menganalisis dan merancang sebuah *platform* virtualisasi berbasis Docker yang dipasang di operasi *system* Linux Ubuntu 17.10, meliputi proses instalasi, konfigurasi, dan menjalankan aplikasi Docker. Docker adalah sebuah projek terbuka yang berfungsi membantu cara kerja untuk *developer* atau sistem administrator untuk membangun, mengemas dan menjalankan aplikasi dimanapun dalam sebuah kontainer. arsitektur menggunakan metode klien *server*, Docker klien, mengirimkan permintaan ke docker daemon untuk membangun, mendistribusikan dan menjalankan kontainer Docker. Keduanya Docker klien dan Docker *daemon* berkomunikasi via *socket* menggunakan *Restful* API. Docker menyediakan alat dan *platform* untuk mengelola siklus hidup pada pada kontainer yaitu developer dan sistem administrator dapat mengembangkan aplikasi dan komponen pendukungnya menggunakan kontainer. Kontainer adalah sebuah wadah menjadi unit untuk mendistribusikan dan menjalankan uji coba aplikasi yang akan siap dipasang.

Kata kunci: Docker , *Client Server*, Kontainer , *Docker Daemon, Socket , Restful API*

1. PENDAHULUAN

Di antara banyaknya masalah di dunia perkembangan aplikasi, ada satu masalah klasik yang sering ditemui oleh pengembang aplikasi dan sistem administrator. Masalah tersebut yaitu perbedaan *platform* sistem operasi yang digunakan para pengembang aplikasi dan sistem administrator, maka diperlukanya rancangan arsitektur platform virtualisasi yang lebih baik dan terdistribusi. Ada yang menggunakan MacOS, Linux ataupun Windows. Variasi *platform* di antara pengguna linux pun akan beragam, baik distro yang digunakan, maupun versi perangkat lunak yang terpasang di dalamnya. Intinya adalah tidak semua pengembang aplikasi menggunakan lingkungan kerja yang sama persis. Teknologi virtualisasi menawarkan perluasan penggunaan pada infrastruktur teknologi informasi menjadi lebih tinggi dengan biaya yang

Diterima 19 Januari 2019, Direvisi 12 Februari 2019, Diterima untuk publikasi 20 Maret 2019

lebih rendah karena virtualisasi memanfaatkan sumber daya komputer secara optimal sehingga dapat digunakan untuk berbagai kebutuhan, setiap kali penambahan aplikasi atau fitur baru maka diperlukan juga menambahkan mesin komputer beserta sistem operasi, sehingga biaya untuk lisensi perangkat lunak makin bertambah dengan virtualisasi penambahan komputer tidak diperlukan karena sebuah server dapat digunakan secara Bersama-sama dengan berbagai aplikasi yang akan diperlukan. Saat ini teknologi virtualisasi kontainer semakin populer terutama docker di mana dapat memasang aplikasi kedalam sebuah kontainer.

Docker adalah sebuah proyek terbuka yang menyediakan *platform* terbuka untuk pengembang aplikasi mau pun sistem administrator untuk dapat membangun, mengemas, dan menjalankan aplikasi di mana pun sebagai sebuah wadah kontainer yang ringan. docker awal mulanya dikembangkan oleh *Solomon hykes* sebagai proyek internal di *dotcloud*, sebuah perusahaan PAAS. Sistem kontainer docker melakukan pembungkusan pada bagian-bagian perangkat lunak dalam berkas sistem secara lengkap yang berisi semua kebutuhan yang diperlukan dalam menjalankan kode, *runtime*, sistem alat dan sistem perpustakaan yang dapat dijalankan pada *server*, sehingga perangkat lunak dapat berjalan sama meskipun dalam lingkungan yang berbeda. Kontainer berjalan pada satu komputer tunggal yang dapat dipasang berbagai sistem operasi dalam *kernel* yang sama dapat mengurangi penggunaan RAM dibangun dengan berkas sistem yang berlapis sehingga penggunaan memori menjadi lebih efisien.

1.1 Rumusan Masalah

Berdasarkan penjelasan di atas pada bagian latar belakang, maka penulis merumuskan berbagai masalah yang akan dianalisis dengan penelitian yang akan penulis yaitu:

1. Bagaimana rancangan dan arsitektur *platform* virtualisasi berbasis Docker?
2. Bagaimana keefektifan rancangan arsitektur Docker sebagai *platform* virtualisasi berbasis kontainer.
3. Bagaiamana membuktikan kompatibilitas kontainer Docker untuk menjalankan aplikasi yang telah berjalan di atas *platform* sistem operasi yang berbeda.
4. Bagaimana docker dapat digunakan untuk membantu proses kontrol versi sebuah aplikasi.

1.2 Tujuan dan Manfaat Penelitian

1.2.1 Tujuan

Tujuan yang ingin dicapai dalam penulisan tugas akhir ini adalah sebagai berikut:

1. Merancang arsitektur *platform* virtualsasi berbasis docker yang mudah digunakan dan cepat dalam implementasi serta lebih terdistribusi.
2. Menguji efektifitas rancangan asitektur docker sebagai platform virtualisaisi berbasis kontainer.
3. Menguji kompatibilitas Docker sebagai platform virtualisasi untuk mengatasi permasalahan terkait ketergantungan lintas *platform*.
4. Melakukan pengujian kontrol versi untuk aplikasi.

1.2.2 Manfaat Penelitian

Dalam penelitian ini manfaat yang didapatkan adalah sebagai berikut:

1. Memahami cara kerja dan penggunaan docker sebagai salah satu *platform* berbasis kontainer.
2. Menghasilkan karya tugas akhir di bidang virtualisasi berbasis Docker yang dapat menjadi salah satu rujukan karya tulis dan tugas akhir yang bermanfaat bagi dunia akademis dan dunia industri.

1.3 Batasan Masalah

Dalam penelitian ini berikut beberapa batasan masalah di antaranya:

1. Perangkat lunak Docker dipasang pada sistem operasi Ubuntu versi 17.10.
2. Manajemen sistem Docker menggunakan alat yang telah tersedia (*built in*) pada perangkat lunak docker dan alat manajemen *rancher server*.
3. Pada uji kompatibilitas Docker untuk mengatasi permasalahan terkait ketergantungan lintas *platform* hanya menguji sebuah aplikasi web yaitu joomla.
4. Dalam penelitian ini tidak dilakukan pengujian keamanan pada sistem docker.

2. METODE PENELITIAN

Dalam tulisan ini penulis menggunakan metode kualitatif deskriptif dan eksperimental dengan menggunakan perangkat lunak virtualisasi dan membuat simulasi yang disesuaikan dengan kasus aslinya. Adapun teknik pengumpulan datanya dilakukan dengan dua unsur pengumpulan data:

1. Studi pustaka data diperoleh dari tulisan-tulisan dan panduan di internet yang berkaitan dengan masalah yang penulis bahas.
2. Simulasi dilakukan dengan percobaan pembuatan pemodelan dengan berdasarkan panduan yang berkaitan dengan masalah yang dibahas.

3. KEBUTUHAN PERANGKAT LUNAK DAN PERANGKAT KERAS

3.1 Kebutuhan Perangkat Lunak

Tabel 1. Kebutuhan Perangkat Lunak

Sistem Operasi/ Aplikasi	Versi	User Interface	Guest/ Host	Penggunaan
Ubuntu	17.10 Mate	Desktop	Host	Sebagai Host utama dan juga sebagai client.
Docker	1.13.1	CLI	Host	Melakukan pull image dan Menjalankan Kontainer
Portainer	-	CLI	Guest/ Host	Uji Kompabilitas terhadap Perbedaan Lingkungan Host
Centos	7	CLI	Guset/ Host	Uji Kompabilitas terhadap Perbedaan Lingkungan Host/ Distro
Virtualbox	-	Desktop	-	Menjalankan Ubuntu 18.04
NFS Server	-	CLI	Host	Backup data dan sharing File
Joomla, moodle, wordpress	-	Web Application	-	Aplikasi yang akan dijalankan dalam bentuk kontainer

3.2 Kebutuhan Perangkat Lunak

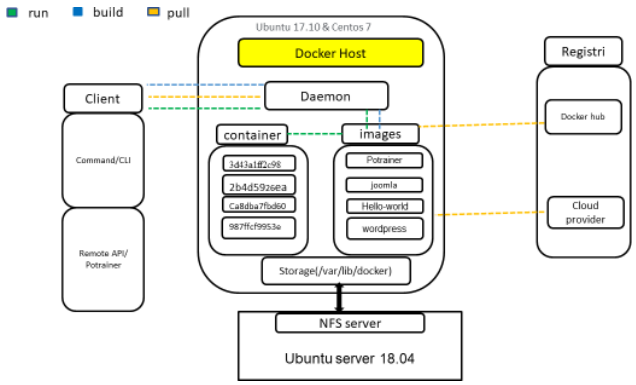
Standard requirement untuk Ubuntu Desktop Edition

1. Processor : 2 GHz *dual core processor*.
2. RAM : 2 GB RAM (*system memory*).
3. Harddisk : 25 GB of *hard-drive space*.
4. VGA : kemampuan resolusi layar 1024x768.
5. CD/DVD atau port USB untuk media installasi.
6. Koneksi internet.

4. PERANCANGAN SISTEM

4.1 Perancangan Arsitektur Sistem

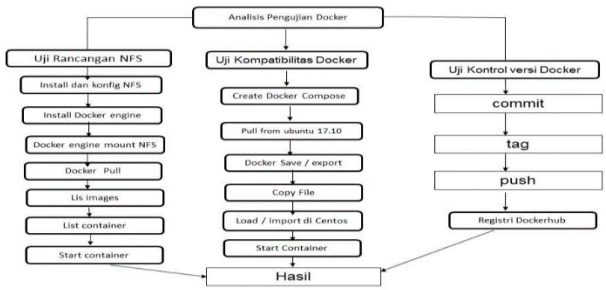
Dalam penelitian ini peneliti akan melakukan rancangan terhadap arsitektur sistem. Arsitektur yang dirancang menggunakan pendekatan rancangan sistem terdistribusi, dimana dilakukan pemisahan antara media penyimpanan (penyimpanan untuk *images* dan file sistem Docker *container*) dari docker engine itu sendiri. Didalam arsirektur umum penyimpanan ini berada dalam satu *host* yang sama dengan Docker *engine*. Namun didalam rancangan arsitektur yang akan dibangun ini, teknik pemisahan media penyimpanan (penyimpanan untuk images dan file sistem docker container) menggunakan solusi *network file system* (NFS). Ilustrasi mengenai rancangan arsitrktur sistem yang akan dibangun di dalam penelitian ini dapat dilihat pada gambar.



Gambar 1. Perancangan Arsitektur Sistem

4.2 Perancangan Pengujian

Pada penelitian ini tiga fokus yang diteliti yaitu penelitian pada efektifitas, uji kontrol versi, dan kompatibilitas docker sebagai *platform* virtualisasi dengan melakukan uji coba secara langsung menggunakan laptop peneliti, kemudian menjalankan docker dan mengakses aplikasi yang telah terpasang. Berikut adalah bagan pengujian yang akan coba penulis rancang.



Gambar 2. Perancangan Pengujian

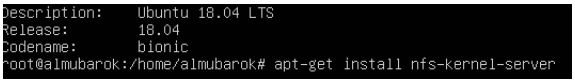
5. IMPLEMENTASI DAN PENGUJIAN

5.1 Pemasangan NFS-server dan NFS-client

Pada pengujian ini peneliti akan melakukan pemasangan NFS-*server* pada ubuntu server 18.04 dan NFS-client pada Ubuntu 17.04 , dan melakukan remote direktori *file system* Docker yaitu /var/lib/docker dengan melakukan *mounting* pada *client*, dengan masing-masing IP sebagai berikut:

```
Ubuntu server 18.04 (host server):
192.168.2.xxx Ubuntu 17.10
(client): 192.168.2.xxx
Centos 7 (client): 192.168.2.xxx
```

a. Langkah pertama yaitu melakukan pemasangan komponen NFS-*server* dan NFS-*client*, Pada host server ubuntu 18.04 akan dipasang NFS-kernel-server yang berfungsi untuk melakukan berbagi direktori dengan menggunakan perintah apt pada terminal:



Gambar 3. Pemasangan Komponen NFS-*server*

b. Kemudian pada *client* ubuntu 17.10 memasang NFS-common dan pada centos NFS-utils yang berfungsi untuk melakukan *remote* folder yang berada pada server. Dengan melakukan perintah:

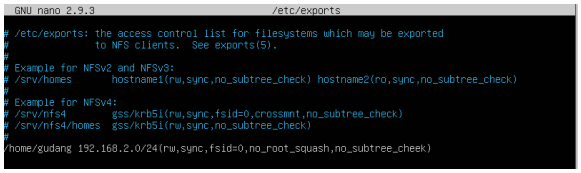
```
root@almubarok:/# sudo apt-get
install nfs-common
```

c. Membuat direktori folder yang akan di-*share* yaitu pada *home* dengan nama folder gudang.

d. Langkah selanjutnya adalah melakukan konfigurasi NFS-*export* pada *host server* dengan mengedit file *exports* pada direktori /etc/ menggunakan *text editor* nano, pada file *exports* tersebut ditambahkan kode yang berisi struktur direktori yang akan di share, alamat ip client yang mengakses, dan hak akses pada folder yang di share, berikut adalah contohnya:

```
root@almubarok:/# nano
```

Dan menambahkan kode di bawah ini:



Gambar 4. Konfigurasi /etc/export

Mengecek ulang apakah akses mounting sudah terdapat dalam list dengan perintah:

```
root@almubarok:/#
```

e. Setelah selesai melakukan pengeditan kemudian di simpan dan jalankan perintah:

```
root@almubarok:/# systemctl
```

f. Pastikan firewall dalam kondisi tidak aktif dengan menjalankan perintah:

```
root@almubarok:#sudo ufw status
Status : inactive
```

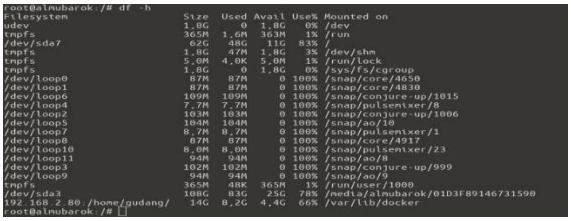
g. Langkah selanjutnya mount point pada klien, setelah host server sudah dapat melakukan sharing. Maka salin terlebih dulu folder *var*/lib/docker pada klient ke folder *mnt*/temp, setelah disalin pada klien, matikan terlebih dahulu docker engine nya dengan perintah:

```
root@almubarok:/# systemctl stop
```

h. Lakukan *mounting* pada klien ke NFS-server dengan perintah:

```
Sudo mount
192.168.2.80:/home/gudang
/var/lib/docker
```

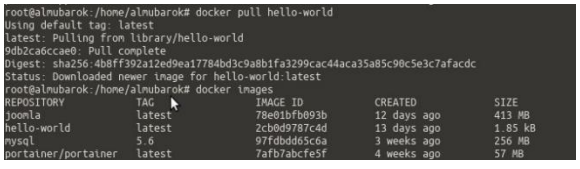
i. Dengan perintah di atas, folder yang telah di-*share* pada *host server* sudah dimounting pada client, dapat dicek dengan menjalankan perintah df –h.



Gambar 5. Menampilkan *Mounting* Direktori

5.2 Pengujian NFS-*server* dan Docker

a. Melakukan uji coba *pull image* pada klien dan menjalankan container serta mengecek apakah ada pertambah size memori pada direktori NFS-server, dengan menjalankan perintah berikut:

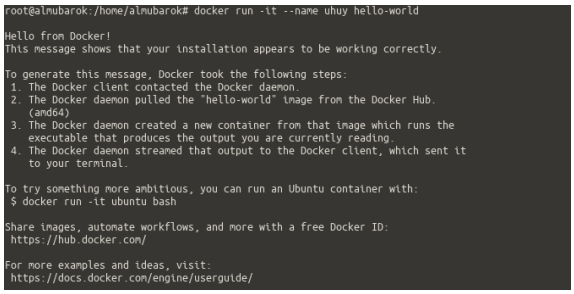


Gambar 6. Uji Coba *Pull Image*

Pada gambar di atas telah diketahui bahwa images hello-world berhasil diunduh. Setelah *pull* berhasil kemudian jalankan kontainer *hello-world* misalkan dengan nama uhuy dengan perintah:

```
root@almubarok:# docker run -it
--name uhuy hello-world
```

Maka container akan menampilkan teks hello--world pada CLI sebagai berikut:



Gambar 7. Tampilan Hello World pada CLI

b. Kemudian cek kembali, apakah file sistem pada NFS-server sudah ada pertambahan size dengan perintah:

```
root@nfserver:#du -sh /home/gudang
```

Gambar 8. Tampilan Hasil Pengecekan pada CLI

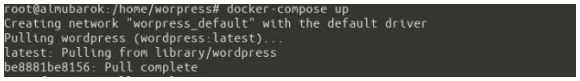
Dari gambar di atas dapat diketahui bahwa *image* yang telah di pull pada klien berhasil terpasang pada NFS-server pada ubuntu server 18.04 dengan bertambahnya size pada direktori /*home/*gudang. Dengan demikian file system Docker engine sudah berhasil terpasang pada NFS-server.

5.3 Uji Kompatibilitas Docker terhadap Perbedaan Sistem Operasi

Docker *engine* yang telah terpasang pada sistem operasi tersebut, berikut adalah langkah-langkah pengujian kompatibilitas docker tehadap perbedaan sistem:

a. Melakukan *pull image* Wordpress pada Ubuntu 17.10 menggunakan docker compose dengan langkah-langkah sebagai berikut:

1. Membuat file docker compose dengan ekstensi, yml, dengan kode program.
2. Selanjutnya menjalankan perintah docker-compose up maka image wordpress otomatis diunduh seperti gambar di bawah ini:



Gambar 9. Docker *compose-up*

Dari gambar diatas dapat diketahui bahwa pull image menggunakan Docker *compose* telah berhasil, dapat dibuktikan dengan perintah selanjutnya.

b. Mengecek apakah images sudah berhasil unduh dengan perintah:

```
root@almubarok:/# docker images
root@almubarok:/# docker
ps -a
```

Jika kontainer dalam keadaan *exited* maka perlu menjalankan Docker *start.*

c. Save image dan exort kontainer wordpress pada ubuntu 17.10 save image wordpress dan mysql:5.7 dengan perintah:

```
root@almubarok:/home# sudo
docker save wordpress >
wordpress1.tar
root@almubarok:/home# sudo
   docker save mysql:5.7 >
   mysql1.tar
```

Kemudian cek pada direktori *home*, hasil dari *save image* yang berektensi wordpress1.tar dan mysql1.tar.salin yang file telah disave dan diexport kedalam folder home yang berada pada sistem operasi centos kemudian jalankan perintah load dan import:

Gambar 10. File Hasil *Save* dan *Export*

d. Kemudian melakuan load image dan cek hasilnya dengan list images.

Gambar 11. Pengecek Hasil *List Image*

Dari gambar diatas dapat diketahui bahwa image yang telah disave pada ubuntu berhasil di-*load* pada sistem operasi centos dan membuktikan bahwa Docker mampu mengatasi masalah perbedaan sistem operasi dalam hal menjalankan aplikasi *(conflicting dependencies).*

5.4 Uji Kontrol Versi Aplikasi pada Joomla

a) Mengecek image id yang akan dilakukan kontrol versi

```
root@almubarok:/# docker images
```

```
root@almubarok:/# docker ps
```

b) Melakukan commit image joomla dengan perintah:

```
root@almubarok:# docker
commit 0319dc55ef03
almubarok95/joomla:07.18
```

c) Melakukan docker tag untuk versioning pada images dengan perintah:

```
root@almubarok:# docker tag
78e01bfb093b
almubarok95/joomla:07.18
```

d) Setelah melakukan tag lalu melakukan push ke docker hub, login terlebih dahulu pada akun *dockerhub*-nya dengan perintah docker login:

```
root@almubarok:# docker
login Username
(almubarok95):almubarok95
Password:
Login Succeeded
```

e) Melakukan push image Joomla Docker dengan perintah docker push:

```
root@almubarok:# docker push
almubarok95/joomla:07.18
```

f) Kemudian melakukan pengecekan dengan login pada situs *dockerhub* melalui browser.

6. KESIMPULAN DAN SARAN

6.1 Kesimpulan

Dengan hasil pengujian pada bab sebelumnya penguji mendapatkan kesimpulan sebagai berikut:

1. Rancangan virtualiasi berbasis Docker yang diuji yaitu menggunakan NFS-*server* memisahkan antara file sistem Docker dengan Docker *engine* dimana Docker *engine* berada pada dua komputer klien dan file sistem Docker tersimpan pada komputer *server* yang telah dipasang NFS-*server*, file sistem Docker yang berada pada direktori /var/lib/docker di-*mounting* pada NFS *server* yang berada pada direktori /home/gudang.
2. Efektifitas rancangan *platform* virtualisasi berbasis docker menggunakan NFS-*server* berjalan dengan baik, serta dalam proses pengembangan aplikasi menjadi lebih terdistribusi, dapat mengatasi masalah jika ada salah satu docker engine yang gagal aktif (*crash*).
3. Kompatibilitas Docker dalam mengatasi perbedaan pada sistem operasi sangat baik itu buktikan dengan melakukan *export* dan *import* kontainer Docker menggunakan sistem operasi ubuntu 17.10 dan centos6.
4. Uji kontrol versi aplikasi menggunakan Docker sangat efesien dan mudah dilakukan dengan waktu relatif lebih singkat.
5. Manajemen Docker menggunakan portainer memberikan kemudahan dalam hal pengelolaan pada setiap komponen-komponen docker yang terpasang.

6.2 Saran

Dalam penulisan ini masih banyak kekurangan dalam implementasinya, berikut adalah saran untuk penelitian selanjutnya:

1. Perlu adanya penelitian lebih lanjut mengenai bagaimana *high availability* pada NFS-*server* dalam menjalankan aplikasi berbasis kontainer.
2. Dalam rancangan ini peneliti menjalankan Docker pada jaringan lokal, Docker dapat dimanfaatkan ketahap *platform multi-cloud* sebagai wadah komputasi awan.

DAFTAR PUSTAKA

[1] Goasguen, Sebastien, "*Docker Cookbook*," United States of America: O'Reilly Media, Inc., 2015.

[2] "Dasar Manajemen Docker menggunakan Portainer," Cilsy Fiolotuion Indonesia, 2017.

[3] R. Mckendrick, S. Gallagher, "*Mastering* Docker," *second edition*, Birmingham: Packtpublishing Ltd., 2017.

[4] V. Sugianto, R. Pratama, "*Virtualisasi Modern berbasis* Docker," Bekasi: Penerbit PT Excellent Infotama Kreasindo, 2016.

[5] Javacodegeeks, "*Docker Containerization Cookbook*," Exelixis Media P.C, 2016.

[6] *"Ubuntu is the numberone for container,"* ubuntu.com 2017, [Online]. Available: https://www.ubuntu.com/containers

[7] James Turnbull, "*The Docker Book: Containerization is a the new Virtualization*," Melbourne, Australia, 2017.