Implementasi Penyimpanan Data Persisten pada Docker Swarm Menggunakan Network File System (NFS)

SKRIPSI

Disusun oleh:
Andreas Frederius
NIM: 135150201111263

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI PENYIMPANAN DATA PERSISTEN PADA DOCKER SWARM MENGGUNAKAN NETWORK FILE SYSTEM (NFS)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Andreas Frederius
NIM: 135150201111263

Skripsi ini telah diuji dan dinyatakan lulus pada
7 Desember 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Mahendra Data, S.Kom., M.Kom	Widhi Yahya, S.Kom., M.Sc.
NIK: 2015038611171001	NIK: 2016078911211001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T, Ph.D
NIP: 197105182003121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 13 Desember 2018

Andreas Frederius

NIM: 135150201111263

ABSTRAK

Docker Swarm merupakan teknologi pengembangan sistem terdistribusi untuk melakukan manajemen pada kelompok mesin Docker. Dengan Docker Swarm dapat menjalankan banyak kontainer sekaligus pada kelompok mesin Docker. Pada penerapan sistem terdistribusi menggunakan Docker Swarm diperlukan sebuah penyimpanan data yang persisten. Namun masalahnya Docker Swarm menyimpan data pada kontainer, jika kontainer terhapus maka data akan ikut terhapus. Maka dari itu diperlukan sebuah alternatif penyimpanan data yang persisten. Penelitian sebelumnya menggunakan Storage Class Memory (SCM). SCM adalah teknologi perangkat keras baru yang menawarkan penyimpanan persisten, dan cepat untuk kontainer. Namun SCM merupakan teknologi perangkat keras yang khusus dan memerlukan biaya yang mahal. Alternatif lain dapat menggunakan Network File System (NFS). NFS merupakan *open protokol* yang dapat digunakan untuk berbagi *file* pada banyak jaringan komputer dan sistem operasi. Perancangan arsitektur NFS pada Docker Swarm menggunakan arsitektur *client-server*. Docker Swarm berperan sebagai *client* dan NFS berperan sebagai *server*. NFS mampu menyediakan penyimpanan data persisten pada Docker Swarm dengan melakukan sinkronisasi data sekalipun kontainer dihapus dan mesin di-*restart*. NFS dapat melakukan sinkronisasi data pada Docker Swarm untuk mengambil data yang telah tersimpan pada NFS sehingga data tetap persisten. Kinerja kecepatan *write* rata-rata pada NFS adalah 30.168 KB sedangkan kinerja kecepatan *read* rata-rata pada NFS adalah 63.939 KB.

Kata kunci: Docker Swarm, Network File System, penyimpanan persisten

ABSTRACT

Docker Swarm is a distributed system development technology for managing the Docker engine group. Docker Swarm can run multiple containers at the same time with Docker engine group. In implementing distributed systems using Docker Swarm, a persistent data storage is required. But the problem is Docker Swarm stores data on the container, if the container is deleted then the data will also be deleted. Therefore we need an alternative persistent data storage. Previous research used Storage Class Memory (SCM). SCM is a new hardware technology that offers persistent, fast storage for containers. But SCM is a hardware technology that is specialized and requires a high cost. Other alternatives can use the Network File System (NFS). NFS is an open protocol that can be used to share files on many computer networks and operating systems. The design of the NFS architecture on Docker Swarm uses a client-server architecture. The Swarm Docker acts as a client and NFS acts as a server. NFS is able to provide persistent data storage on Docker Swarm by synchronizing data even if the container is deleted and the machine is restarted. NFS can synchronize data on Docker Swarm to retrieve data that has been stored on NFS so that the data remains persistent. The average write speed performance on NFS is 30,168 KB while the average read speed performance on NFS is 63,939 KB.

Keywords: Docker Swarm, Network File System, *persistent storage*

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa karena berkat rahmat dan anugrah-Nya, penulis bisa menyelesaikan laporan tugas akhir skripsi dengan judul Implementasi Penyimpanan Data Persisten pada Docker Swarm menggunakan Network File System.

Dengan selesainya laporan tugas akhir skirpsi ini, tidak terlepas dari dukungan teman-teman dan dosen, maka penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya penulis mampu menyelesaikan laporan tugas akhir skripsi ini dengan baik,
2. Ayahanda dan Ibunda dan seluruh keluarga besar atas segala nasihat, kasih sayang, perhatian dan kesabarannya di dalam membesarkan dan mendidik penulis, serta yang senantiasa tiada henti-hentinya memberikan doa dan semangat demi terselesaikannya skripsi ini,
3. Bapak Agus Wahyu Widodo, S.T., M.T., M.Sc. selaku ketua Program Studi Teknik Informatika Universitas Brawijaya,
4. Bapak Tri Astoto Kurniawan, S.T., M.T., ph.D selaku Ketua Jurusan Teknik Informatika Universitas Brawijaya,
5. Bapak Mahendra Data, S.Kom., M.Kom selaku dosen pembimbing I, dan Bapak Widhi Yahya, S.Kom., M.Sc. selaku dosen pembimbing II yang telah dengan sabar membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga saran dan kritik yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat membawa manfaat bagi semua pihak yang menggunakannya.

Malang, 13 Desember 2018

Penulis

andreas_fds@yahoo.com

DAFTAR GAMBAR

Gambar 2.1 Arsitektur Virtualisasi Kontainer dan Mesin Virtual 5
Gambar 2.2 Arsitektur Docker 7
Gambar 2.3 Docker Swarm 8
Gambar 2.4 Arsitektur NFS 9
Gambar 2.5 Komunikasi NFS 10
Gambar 3.1 Diagram Alir Metodologi Penelitian 11
Gambar 3.2 Diagram Alir Perancangan Sistem pada Docker Swarm 14
Gambar 3.3 Diagram Alir Perancangan Sistem pada NFS.....15
Gambar 3.4 Perancangan Arsitektur Sistem 15
Gambar 3.5 Perancangan Arsitektur pada Docker Swarm 16
Gambar 3.6 Perancangan Arsitektur pada NFS 17
Gambar 3.7 Perancangan Topologi Sistem 17
Gambar 4.1 Implementasi Lingkungan Sistem 20
Gambar 4.2 Tampilan VMWare 20
Gambar 4.3 Mesin Virtual Manager 21
Gambar 4.4 Mesin Virtual Worker 21
Gambar 4.5 Mesin Virtual NFS Server 22
Gambar 4.6 Diagram Alir Implementasi NFS 22
Gambar 4.7 NFS Berjalan 23
Gambar 4.8 Diagram Alir Implementasi Docker Swarm 24
Gambar 4.9 Inisiasi Swarm Manager 25
Gambar 4.10 Swarm Worker gabung 25
Gambar 4.11 Docker Volume Plugin Berjalan pada Swarm Manager 25
Gambar 4.12 Docker Volume Plugin Berjalan pada Swarm Worker 26
Gambar 4.13 Menjalankan Docker Compose 27
Gambar 5.1 Kontainer yang Berjalan 30
Gambar 5.2 Tampilan Upload Gambar pada Wordpress 30
Gambar 5.3 Tampilan *Posting* pada Wordpress 31
Gambar 5.4 Tampilan Kontainer dihapus 31

Gambar 5.5 Tampilan Kontainer Membuat dan Menjalankan Kontainer Baru 32
Gambar 5.6 Data Kontainer Wordpress dan MySQL yang Tersimpan pada NFS ...32
Gambar 5.7 Tampilan data *Upload* yang Tersimpan pada NFS 33
Gambar 5.8 Tampilan *Posting* pada Wordpress 33
Gambar 5.9 Grafik Kecepatan *Read-Write* per Detik 36
Gambar 5.10 Jumlah Operasi per Detik 37
Gambar 5.11 Jumlah KB *Read-Write* per Operasi 37
Gambar 5.12 Jumlah Waktu Rata-Rata RTT 38

DAFTAR TABEL

Tabel 2.1 Penelitian terkait 4
Tabel 2.2 Pebandingan fitur mesin virtual dan kontainer 6
Tabel 3.1 Minimum System Requirement VMware Workstation 12 Pro 13
Tabel 3.2 Minimum System Requirement Docker Swarm 13
Tabel 3.3 Minimum System Requirement NFS 13
Tabel 4.1 Spesifikasi laptop 19
Tabel 4.2 Pengaturan Mesin Virtual Docker Swarm 19
Tabel 4.3 Pengaturan Mesin Virtual NFS Server 19
Tabel 4.4 Output Nfsiostat 28
Tabel 5.1 Pengujian Fungsional 34
Tabel 5.2 Output *Write* NFS 34
Tabel 5.3 Output *Read* NFS 34
Tabel 5.4 Output *Read* dan *Write* Mesin Virtual 35

DAFTAR ISI

DAFTAR ISI vii
DAFTAR TABEL ix
DAFTAR GAMBAR x
DAFTAR LAMPIRAN xii
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan Masalah 2
1.3 Tujuan 2
1.4 Manfaat 2
1.5 Batasan masalah 2
1.6 Sistematika Pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN
2.1 Penelitian Terkait 4
2.2 Docker 5
2.2.1 Arsitektur Docker 6
2.2.2 Docker Compose 7
2.2.3 Docker Swarm 8
2.2.4 Docker Volume Plugin 9
2.3 NFS 9
2.3.1 Arsitektur NFS 9
BAB 3 METODOLOGI
3.1 Identifikasi Masalah 12
3.2 Studi Literatur 12
3.3 Analisis Kebutuhan 12
3.3.1 Kebutuhan Fungsional 12
3.3.2 Kebutuhan Non Fungsional 12
3.4 Perancangan Sistem 13
3.4.1 Perancangan Sistem pada Docker Swarm 14
3.4.2 Perancangan Sistem pada NFS 14
3.4.3 Perancangan Arsitektur Sistem 15

3.4.4 Perancangan Arsitektur Docker Swarm 16
3.4.5 Perancangan Arsitektur NFS 17
3.4.6 Perancangan Topologi Sistem 17
3.5 Implementasi dan Pengujian.....18
3.6 Hasil dan Analisis 18
3.7 Penarikan Kesimpulan 18
BAB 4 IMPLEMENTASI DAN PENGUJIAN
4.1 Spesifikasi Sistem 19
4.2 Implementasi Sistem 20
4.2.1 NFS 22
4.2.2 Docker Swarm 24
4.3 Pengujian 27
4.3.1 Pengujian Fungsional 28
4.3.1.1 Pengujian Persistensi NFS 28
4.3.2 Pengujian Fungsional 28
4.3.2.1 Pengujian Kinerja Kecepatan *Write* pada NFS 29
4.3.2.2 Pengujian Kinerja Kecepatan *Read* pada NFS 29
4.3.2.1 Pengujian Kinerja Kecepatan *Read* dan *Write* pada NFS 29
BAB 5 HASIL DAN ANALISIS
5.1 Hasil Pengujian Fungsional 30
5.1.1 Hasil Pengujian Persistensi NFS 30
5.2 Hasil Pengujian Non Fungsional 34
5.2.1 Hasil Pengujian Kinerja Kecepatan *Write* pada NFS 34
5.2.2 Hasil Pengujian Kinerja Kecepatan *Read* pada Mesin Virtual 34
5.2.3 Hasil Pengujian Kinerja Kecepatan *Read* dan *Write* pada Mesin Virtual.....35
5.3 Analisis Pengujian Fungsional 35
5.3.1 Analisis Persistensi NFS 35
5.4 Analisis Pengujian Non Fungsional 35
5.4.1 Analisis Perbandingan Kinerja Read dan Write NFS dengan Mesin Virtual 35

BAB 6 PENUTUP

6.1 Kesimpulan 39

6.2 Saran 39

DAFTAR PUSTAKA 40

LAMPIRAN 42

BAB 1 PENDAHULUAN

1.1 Latar belakang

Docker merupakan salah satu produk untuk mewujudkan virtualisasi berbasis kontainer. Docker menjadi alternatif virtualisasi yang ringan untuk pengembangan aplikasi dan juga popular di dalam industri perangkat lunak (Naik, 2016). Kontainer adalah sebuah abstraksi pada layer aplikasi yang membungkus paket-paket perangkat lunak dan semua yang diperlukan untuk menjalankannya (Docker, 2018). Kontainer berbeda dengan virtualisasi yang memerlukan sistem operasi tersendiri untuk masing-masing mesin virtual, sedangkan banyak kontainer dapat berbagi sistem operasi yang sama (Dua, 2014). Kontainer memiliki beberapa keunggulan seperti kinerja yang cepat, dan efisiensi sumber daya dibanding dengan mesin virtual (Lee, 2017).

Docker memperkenalkan teknologi pengembangan sistem terdistribusi yang di sebut Swarm. Docker Swarm adalah sebuah solusi yang diberikan Docker untuk melakukan manajemen sebuah kelompok Docker yang berjalan pada banyak mesin yang disebut Swarm (Nguyen, 2017). Secara *default* Docker memungkinkan untuk menyimpan data dalam kontainer. Namun ketika kontainer terhapus maka data pada kontainer akan ikut terhapus. Data pada kontainer juga tidak bisa dipindahkan atau digunakan untuk saling berbagi data (Docker, 2018). Oleh karena itu diperlukan sebuah penyimpanan data persisten eksternal untuk memastikan data pada kontainer tetap ada setelah kontainer dihapus. Dalam hal ini Docker menyediakan Docker Volume Plugin untuk mengakses penyimpanan data persisten eksternal (Docker, 2018).

Penelitian sebelumnya yang dilakukan oleh Giles terkait penyimpanan data persisten pada kontainer Docker menggunakan Storage Class Memory (SCM). SCM adalah teknologi perangkat keras baru yang menawarkan penyimpanan persisten, cepat, dan *byte-addressable* untuk kontainer. Kontainer menggunakan *driver* Containerized Storage Class Memory (CSCM) untuk melakukan *read* dan *write* data pada SCM (Giles, 2016). Namun SCM merupakan teknologi perangkat keras yang khusus dan memerlukan biaya yang mahal maka diperlukan alternatif lain untuk penyimpanan data persisten pada Docker.

Berdasarkan latar belakang yang telah diuraikan di atas, maka diperlukan alternatif lain untuk penyimpanan data persisten pada Docker Swarm. NFS adalah sebuah protokol yang dikembangkan oleh Sun Microsystem pada tahun 1984. NFS merupakan *open protokol* yang didefinisikan dalam RFC, mengijinkan siapa pun untuk mengimplementasikan protokol sehingga NFS banyak digunakan secara umum untuk berbagi *file* pada banyak jaringan komputer dan sistem operasi. NFS mengijinkan pengguna mengakses *file* melalui jaringan seperti mengakses *file* di *disk* lokal.

Dalam penelitian ini penulis akan menggunakan NFS sebagai penyimpanan data persisten pada Docker Swarm karena NFS dapat memberikan akses penyimpanan lokal untuk berbagi sumber daya *file* dan penyimpanan. NFS juga

menyediakan transparansi akses *remote* pada *file* sistem, NFS didesain untuk mudah diimplementasikan pada banyak sistem operasi dan arsitektur komputer (Kulkarni, 2010). Dengan menggunakan NFS sebagai penyimpanan eksternal pada Docker Swarm diharapkan dapat memberikan penyimpanan data yang persisten sekalipun kontainer dihapus dan mesin di-*restart*.

1.2 Rumusan masalah

Berdasarkan latarbelakang yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah:

1. Bagaimana perancangan arsitektur penyimpanan data persisten pada Docker Swarm menggunakan NFS?
2. Bagaimana persistensi NFS pada Docker Swarm?
3. Bagaimana kinerja kecepatan *write* data NFS sebagai penyimpanan data persisten pada Docker Swarm?

1.3 Tujuan

Tujuan dari penelitian ini adalah:

1. Merancang dan mengimplementasikan penyimpanan data persisten pada Docker Swarm dengan NFS
2. Mengetahui kinerja kecepatan *read* dan *write* data NFS sebagai penyimpanan data persisten pada Docker Swarm
3. Mengetahui persistensi NFS pada Docker Swarm

1.4 Manfaat

Penelitian ini diharapkan menjadi solusi untuk penyimpanan data persisten pada Docker Swarm menggunakan penyimpanan lokal NFS selain daripada menggunakan vendor penyimpanan *cloud* yang berbayar. Selain itu mampu mengatasi *sharing* data antar kontainer. Adapun manfaat bagi pengguna yaitu menjadi referensi penelitian terkait penyimpanan data persisten pada Docker Swarm.

1.5 Batasan masalah

Berdasarkan latar belakang dan rumusan masalah yang telah diuraikan, agar permasalahan yang dirumuskan dapat lebih fokus, maka penelitian ini dibatasi dalam hal :

1. Konten Docker Swarm yang digunakan untuk pengujian adalah Wordpress dan MySQL
2. Penyimpanan persisten yang digunakan adalah penyimpanan terdistribusi berbasis penyimpanan lokal seperti NFS

1.6 Sistematika pembahasan

Untuk mencapai tujuan yang diharapkan, maka sistematika penulisan yang disusun dalam laporan tugas akhir ini sebagai berikut:

BAB 1 PENDAHULUAN

Pendahuluan menjelaskan tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan dan manfaat dari penelitian serta sistematika penulisan skripsi.

BAB 2 LANDASAN KEPUSTAKAAN

Landasan kepustakaan menguraikan tentang penelitian terkait tentang penyimpanan data persisten pada Docker, dasar teori dan referensi pendukung yang berkaitan dengan Docker Swarm, dan NFS.

BAB 3 METODOLOGI

Metodologi membahas dan menjelaskan langkah-langkah penelitian dan strategi dalam melaksanakan penelitian. Membahas mengenai analisis kebutuhan sistem dan perancangan sistem secara terperinci untuk diterapkan pada implementasi. Perancangan juga menunjukan langkah-langkah untuk mengimplementasikan sistem secara sistematis.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Implementasi membahas mengenai tahap-tahap dalam mengimplementasikan penyimpanan data persisten pada Docker Swarm menggunakan NFS. Implementasi dilakukan berdasarkan perancangan sistem yang telah dibuat.

Pengujian membahas mengenai tentang teknik pengambilan data dan bagaimana scenario pengujian.

BAB 5 HASIL DAN ANALISIS

Menjelaskan hasil analisis dari data hasil pengujian penyimpanan data persisten pada Docker Swarm menggunakan NFS.

BAB 6 PENUTUP

Memuat kesimpulan yang diperoleh dari implementasi dan pengujian jaringan yang dikembangkan serta saran-saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Penelitian Terkait

Penelitian sebelumnya yang dilakukan oleh Giles, menggunakan Storage Class Memory (SCM) sebagai penyimpanan data persisten untuk kontainer Docker. SCM menawarkan penyimpanan yang persisten, cepat, dan *byte-addressable* untuk aplikasi yang berjalan pada kontainer. Kontainer menggunakan *driver* Containerized Storage Class Memory (CSCM) untuk melakukan *read* dan *write* data pada SCM (Giles, 2016). Penelitian ini memiliki kesamaan yaitu mengatasi masalah penyimpanan data persisten pada Docker dengan menggunakan penyimpanan eksternal. Namun SCM merupakan teknologi perangkat keras khusus yang menggantikan *hard disk* lokal sehingga memerlukan biaya tambahan yang mahal.

Penelitian yang terkait selanjutnya oleh Mohamed yang mengatasi masalah bagaimana menambahkan penyimpanan yang menawarkan penyimpanan persisten untuk kontainer Docker. Dalam penelitiannya, Mohamed memperkenalkan Ubiquity Framework yang memungkin kontainer Docker untuk mengakses penyimpanan data persisten. Ubiquity adalah sebuah *service* yang menghubungkan kontainer Docker dengan sistem penyimpanan persisten. Sementara para vendor penyimpanan persisten menawarkan solusi untuk Docker, Ubiquity dibuat bukan hanya untuk kontainer Docker tetapi untuk semua basis kontainer seperti CloudFoundry, Openshift, Kubernet, dll (Mohamed, 2017). Namun Ubiquity hanya merupakan framework yang menghubungkan Docker dengan penyimpanan data persisten.

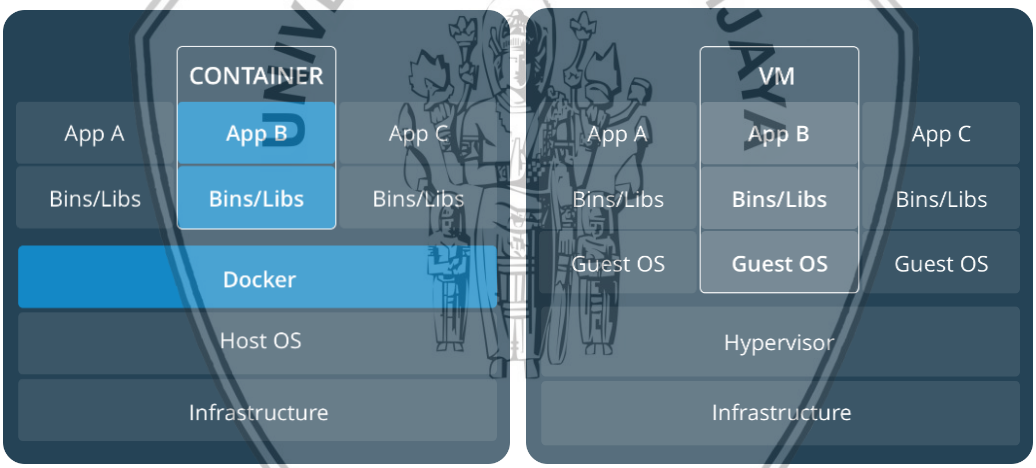
Tabel 2.1 Penelitian terkait

No.	Nama	Tahun	Judul	Variabel	Hasil
1.	Giles	2016	*Container-Based Virtualization for Byte-Addressable NVM Data Storage*	Penyimpanan persisten pada kontainer Docker menggunakan SCM	Throughput memori SCM lebih cepat dengan driver CSCM daripada melalui Docker Volume
2.	Mohamed, dkk.	2017	Ubiquity: Extensible Persistence as a Service for Heterogeneous Container-based Frameworks	Akses penyimpanan persisten untuk vendor kontainer yang berbeda	Ubiquity mendukung penyimpanan persisten untuk CloudFoundry, Kubernet, dan Docker

2.2 Docker

Docker adalah sebuah *open-source engine* yang mengotomatiskan pengembangan aplikasi-aplikasi perangkat lunak ke dalam sebuah kontainer. Docker menyediakan sebuah kontainer yang terisolasi, setiap kontainer memiliki jaringan, penyimpanan, *file* sistem, dan kemampuan untuk mengatur sumber daya dengan banyak kontainer dalam sebuah *host* (Turnbull, 2014). Untuk memahami ide mengenai kontainer dengan mudah dapat dianalogikan seperti sebuah kapal kontainer dimana kontainer-kontainer akan diangkut ke atas kapal kemudian dikirim ke beberapa lokasi berbeda, kontainer pada dasarnya sama yaitu dimana kita dapat membuat aplikasi, dengan mengemasnya di dalam kontainer dan kemudian dijalankan (Nguyen et al., 2016).

Gambar 2.1 menunjukan perbandingan arsitektur virtualisasi antara kontainer dengan mesin virtual. Seperti yang terlihat pada gambar 2.1 semua kontainer berbagi sistem operasi dan *kernel* yang sama sehingga menghemat pemakainaan sumber daya, sedangkan setiap mesin virtual memiliki sistem operasi masing-masing sehingga konsumsi sumber daya lebih besar dan dapat menyebabkan *overhead* (Dua, et al., 2014).



Gambar 2.1 Arsitektur Virtualisasi Kontainer dan Mesin Virtual (Docker, 2018)

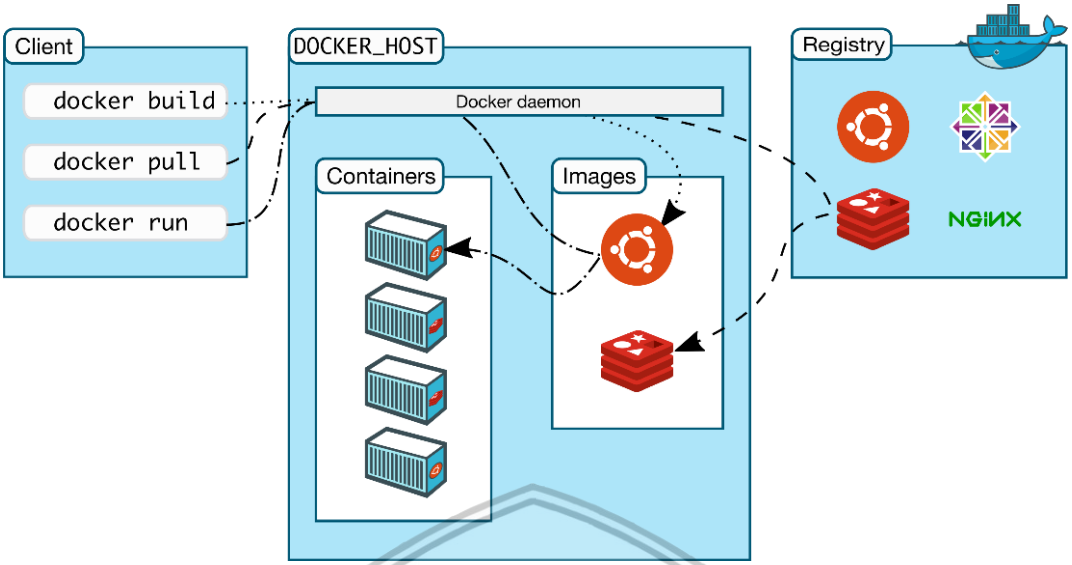
Tabel 2.1 menunjukan perbandingan-perbandingan antara mesin virtual dan kontainer dalam banyak faktor seperti, komunikasi, kinerja, isolasi, jaringan dan penyimpanan.

Tabel 2.2 Pebandingan fitur mesin virtual dan kontainer (Dua, et al.,2014)

Parameter	Mesin virtual	Kontainer
Sistem Operasi	Setiap mesin virtual memiliki sistem operasi masing-masing	Semua kontainer berbagi sistem operasi dan kernel yang sama
Komunikasi	Melalui perangkat Ethernet	Dengan IPC, soket, dll.
Kinerja	Mesin virtual akan mengalami *overhead* ketika instruksi mesin ditranslasi dari Guest OS ke Host OS	Kontainer menyediakan penggunaan sumber daya tanpa *overhead*
Isolasi	Antar Guest OS tidak dapat berbagi *libraries*, *file*, dll.	Sub direktori dapat secara transparan digunakan dan berbagi antar kontainer
Waktu startup	Mesin virtual perlu waktu beberapa menit untuk *boot up*	Kontainer dapat *boot up* dalam waktu beberapa detik
Penyimpanan	Mesin Virtual menggunakan banyak penyimpanan karena semua sistem operasi dan program harus di install	Kontainer menggunakan lebih sedikit penyimpanan karena berbagi sistem operasi dan aplikasi

2.2.1 Arsitektur Docker

Docker menggunakan arsitektur klien-server yang terdiri dari dua komponen: Docker Daemon, dan Docker Client. Docker Daemon mendengarkan permintaan dari Docker Client dan mengatur Docker Objects seperti images, containers, networks, dan volumes. Docker Client menyediakan CLI (*Command Line Interface*) untuk mengirim atau menerima *request* kepada dan dari Docker Daemon (Huang et al., 2017). Docker Registry menyimpan Docker Images yang berisi sebuah aplikasi atau servis. Kontainer adalah sebuah instansi yang dapat dijalankan dari sebuah *image*.



Gambar 2.2 Arsitektur Docker (Docker, 2018)

2.2.2 Docker Compose

Docker Compose adalah sebuah layanan untuk mendefinikan dan menjalankan banyak kontainer. Dengan Docker Compose, dapat membuat sebuah *file* YML untuk mengkonfigurasi layanan-layanan aplikasi hanya dengan sebuah perintah. Docker Compose dapat membuat dan menjalankan semua servis yang sudah didefinisikan dan dikonfigurasi pada *file* YML (Docker, 2018).

Untuk mendefinisikan servis di dalam *file* Compose, dapat membuat sebuah *file* dengan nama docker-compose.yml dengan format pada kode sumber di bawah:

Docker Compose

```
1   version: '3'
2
3   services:
4     web:
5       image: redis:alpine
6       deploy:
7         replicas:6
8         restart_policy:
9           condition: on-failure
10  volumes:
11    datavolume:/redis
```

Services berisi konfigurasi untuk mendefinisikan servis yang berjalan pada setiap kontainer. Image merupakan sebuah servis atau aplikasi yang akan dijalankan. Deploy merupakan spesifikasi konfigurasi untuk membangun dan menjalankan servis-servis. Deploy hanya digunakan ketika membangun servis pada Swarm. Restart_Policy merupakan konfigurasi bagaimana menjalankan

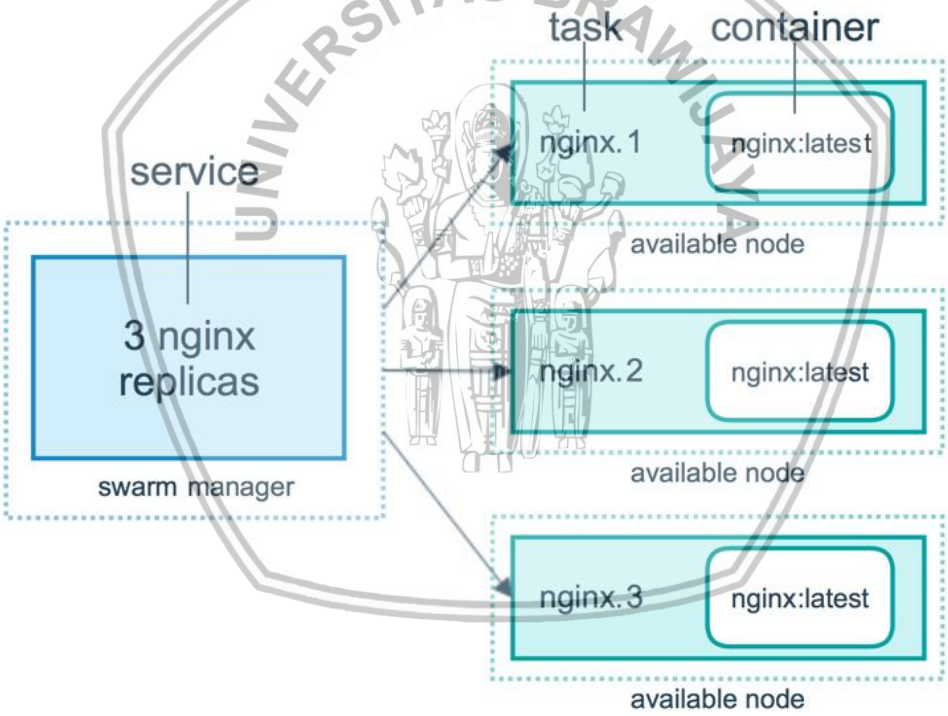
ulang kontainer apabila kontainer berhenti. Volumes merupakan pendefinisian penyimpanan yang akan digunakan pada sebuah servis (Docker, 2018).

2.2.3 Docker Swarm

Docker Swarm adalah sebuah mode pada Docker untuk mengelola sekelompok Docker Engine. Docker Swarm memilki dua komponen: Swarm Manager dan Swarm Worker. SwarmManager menangani *request* dari Swarm Worker dan mengatur Swarm Worker. Swarm Worker menjalankan dan memastikan kontainer berjalan (Huang et al., 2017). Menginisialisasi sebuah Swarm menjadi sebuah Swarm Manager untuk mengelola sekelompok Swarm Worker dapat menggunakan perintah seperti contoh di bawah.

```
$ docker swarm init
```

Docker Swarm init akan menghasilkan sebuah token acak yang dapat digunakan oleh Swarm Worker untuk bergabung dengan Swarm Manager (Docker,2018).



Gambar 2.3 Docker Swarm (Docker, 2018)

Gambar 2.3 merupakan gambaran kerja Docker Swarm. Swarm Manager terhubung dengan tiga Swarm Worker. Ketika Swarm Manager menjalan servis yaitu untuk membuat tiga buah replika *nginx,* maka Swarm Manager akan membuat tiga buah tugas yang akan dibagikan kepada Swarm Worker dengan merata.

2.2.4 Docker Volume Plugin

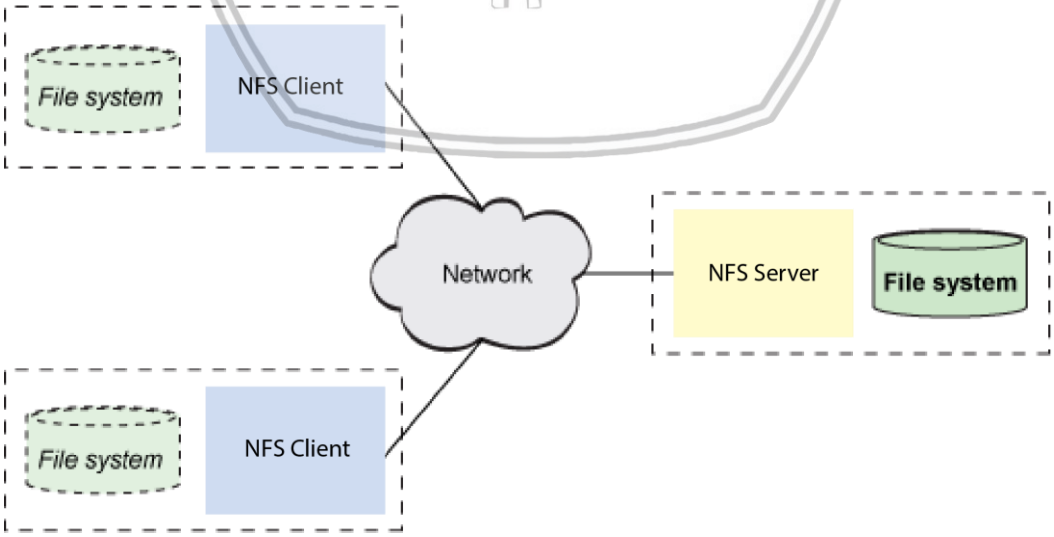
Docker Volume Plugin memungkinkan Docker untuk diintegrasikan dengan sistem penyimpanan eksternal seperti Amazon, sehingga Docker Volume dapat persisten. Docker Volume adalah penyimpanan data pada Docker yang digunakan untuk berbagi data antar kontainer dan host (Giles, 2016). Docker perlu membuat sebuah volume dengan nama yang spesifik untuk diintegrasikan dengan Plugin ke penyimpanan eksternal. Untuk memilih dan menginstal Plugin yang sesuai tersedia dalam dokumentasi Docker (Docker, 2018).

2.3 NFS

NFS (Network File System) dikembangkan oleh Sun Microsystems Inc. tahun 1985 untuk menyediakan *file* sistem terdistribusi di dalam jaringan yang heterogen. NFS menyediakan tranparansi dan akses *remote* untuk berbagi sebuah direktori dan *file* dalam sebuah jaringan komputer seperti mengakses penyimpanan *file* lokal. Di dalam sebuah jaringan NFS memberikan beberapa keuntungan yaitu penggunaan penyimpanan yang lebih sedikit karena data dapat disimpan pada sebuah mesin dan tetap dapat dibagi dan diakses oleh mesin lain. Selain itu NFS juga dapat mengatasi masalah keterbatasan kapasitas penyimpanan karena kapasitas penyimpanan NFS dapat ditambah dengan menambah kapasitas penyimpanan perangkat keras (Canonical, 2018).

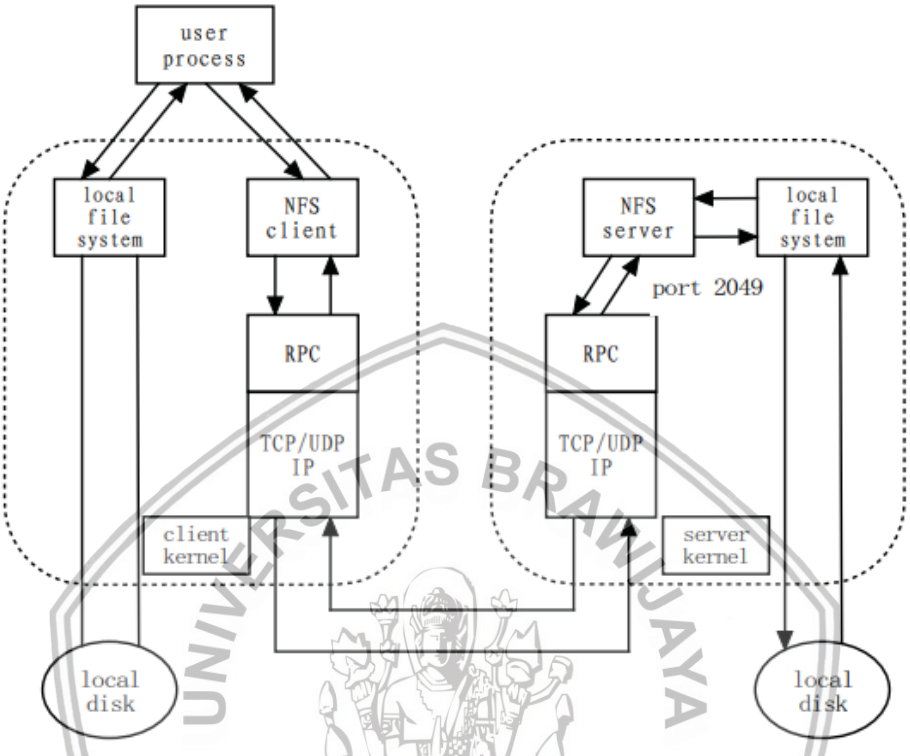
2.3.1 Arsitektur NFS

NFS mengikuti model *client-server*, NFS Server berbagi direktori untuk Client yang terhubung di dalam jaringan. NFS Server bekerja dengan mengeskpor sebuah *file* sistem untuk Client. Client dapat melakukan *mount* pada *file* sistem yang telah diekspor seperti pada *file* sistem lokal (Guo et al., 2013). Arsitektur NFS dapat dilihat pada Gambar 2.4.



Gambar 2.4 Arsitektur NFS (Jones, 2010)

Komunikasi antara NFS Server dengan Client ditangani oleh RPC (Remote Procedure Call) yang berjalan di atas protokol jaringan TCP/ IP. Komunikasi antara NFS Server dan Client ditunjukan pada Gambar 2.5



Gambar 2.5 Komunikasi NFS (Guo, 2013)

NFS client menafsirkan permintaan I/O dan menerjemakannya ke dalam prosedur NFS (OPEN, ACCESS, CREATE, READ, CLOSE, REMOVE dan yang lainnya). Kemudian RPC menyediakan sarana untuk melakukan *procedure call* antar sistem. RPC menyusun permintaan NFS dan argument bersamaan, mengatur pengiriman ke *remote peer* yang tepat, dan kemudian mengatur dan melacak *respons*. Selanjutnya, RPC menggunakan External Data Representation (XDR), yang memastikan bahwa semua NFS menggunakan bahasa atau tipe data yang sama. XDR menangani konversi ke representasi umum (XDR) sehingga semua arsitektur dapat saling beroperasi dan berbagi *file* sistem. Setelah XDR menerjemahkan data ke dalam representasi umum, permintaan ditransfer melalui jaringan pada protokol lapisan transport TCP/IP. Awalnya NFS menggunakan Universal Datagram Protokol (UDP), tetapi saat ini TCP secara umum digunakan karena keandalaannya (Jones, 2010).

Pada server, NFS beroperasi dengan cara yang sama dengan Client. Permintaan mengalir ke jaringan, melalui RPC/XDR, dan ke NFS Server. NFS Server bertanggungjawab untuk memenuhi permintaan dan mendapatkan *file* sistem di dalam lokal *disk*. (Jones , 2010).

BAB 3 METODOLOGI

Bab ini menjelaskan metodologi penelitian yang bersifat implementatif, peneliti akan mengimplementasikan penyimpanan data persistent pada Docker Swarm menggunakan Network File System (NFS). Penelitian ini mulai dengan studi literatur, ruang lingkup jaringan, perancangan sistem, implementasi sistem berdasarkan rancangan yang telah dibuat, serta pengujian, analisis dan penarikan kesimpulan.

Gambar 3.1 Diagram Alir Metodologi Penelitian

3.1 Identifikasi Masalah

Identifikasi masalah dalam penelitian ini adalah bagaimana merancang dan mengimplementasikan penyimpanan data persisten pada Docker Swarm menggunakan NFS. Bagaimana membangun sebuah sistem terdistribusi yang memerlukan penyimpanan persisten menggunakan Docker Swarm. Bagaimana migrasi dan sinkronisasi data kontainer dengan NFS agar data tetap persisten walaupun kontainer terhapus.

3.2 Studi Literatur

Studi literatur diperlukan untuk dapat mendalami permasalahan yang diteliti dan menambah pengetahuan yang diperlukan dalam mengerjakan penelitian dan penulisan laporan skripsi. Dalam hal ini maka peneliti memerlukan informasi-informasi dan solusi terkait permasalahan tersebut. Adapun yang perlu menjadi bahan studi literatur adalah dasar-dasar teori mengenai Docker Swarm, Network System File, dan data persistent.

3.3 Analisis Kebutuhan Sistem

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan dalam implementasi penyimpanan data persisten pada Docker Swarm menggunakan NFS. Menurut Kamus Besar Bahasa Indonesia kata persisten memiliki arti berkesinambungan, terus-menerus, oleh karena itu penyimpanan data persisten pada Docker Swarm dapat diartikan bagaimana memastikan data yang disimpan pada kontainer terus-menerus ada sekalipun kontainer terhapus. Seperti yang dijelaskan pada studi literatur bahwa secara *default* Docker menyimpan data pada kontainer dan data pada kontainer akan ikut terhapus apabila kontainer dihapus. Karena itu untuk mencegah data pada kontainer terhapus diperlukan penyimpanan eksternal diluar kontainer menggunakan NFS.

3.3.1 Kebutuhan Fungsional

Dari penjelasan tersebut, maka penulis melakukan analisis kebutuhan fungsional dari sistem tersebut. Beberapa kebutuhan fungsional dari sistem tersebut yaitu:

1. Docker Swarm dapat menjalankan kontainer
2. Docker Swarm dapat melakukan *read* dan *write* data pada NFS
3. NFS dapat menyimpan data kontianer
4. NFS dapat melakukan sinkronisasi data pada kontainer Docker Swarm

3.3.1 Kebutuhan Non-Fungsional

Dalam penelitian ini, Docker Swarm dan NFS akan dibangun menggunakan perangkat lunak mesin virtual yaitu VMware Workstation 12 Pro. Minimum *system requirement* yang diperlukan untuk menggunakan VMware Workstation 12 Pro adalah sebagai berikut:

Table 3.1 Minimum System Requirement VMware Workstation 12 Pro

Processor	Intel/AMD 2011 ke atas, 1.3GHz atau lebih
Memory	4GB RAM
Disk	1.2 GB ruang kosong
OS	64-bit Windows 7,8,10

Docker Swarm akan dibangun pada sebuah mesin virtual dengan menggunakan sistem operasi ubuntu. Minimum system requirement yang diperlukan untuk Docker adalah sebagai berikut:

Table 3.2 Minimum System Requirement Docker Swarm

Processor	Dual core, 2GHz
Memory	2 GB RAM
Disk	2 GB ruang kosong
OS	64-bit Ubuntu 16.04
Docker version	1.13 ke atas
NFS version	4

Table 3.3 Minimum System Requirement NFS

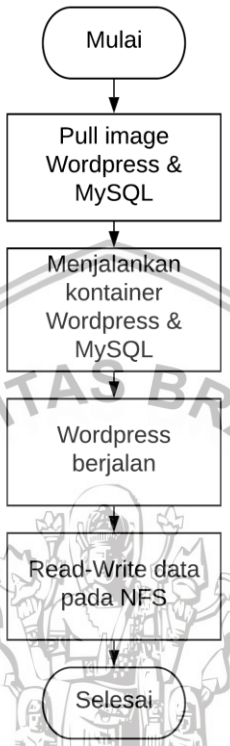
Processor	Dual core, 2GHz
Memory	2 GB RAM
Disk	2 GB ruang kosong
OS	64-bit Ubuntu 16.04
NFS version	4

3.4 Perancangan Sistem

Perancangan sistem dilakukan agar penelitian dapat berjalan dengan sistematis. Perancangan pada penelitian ini meliputi beberapa tahapan, yaitu:

3.4.1 Perancangan Sistem pada Docker Swarm

Docker Swarm akan membangun kontainer Wordpress dan MySQL. Wordpress akan dapat diakses oleh *user* melalui jaringan. Wordpress menyediakan fitur untuk melakukan upload gambar, posting, edit, dan penghapusan. Gambar 3.2 merupakan diagram alir dari alur kerja sistem pada Docker Swarm.

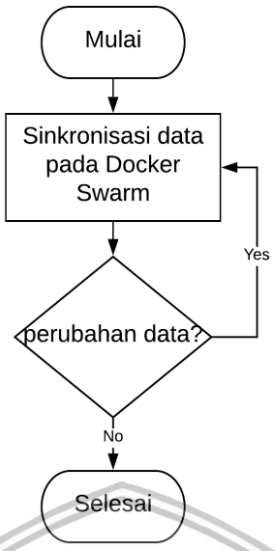


Gambar 3.2 Diagram Alir Perancangan Sistem pada Docker Swarm

Pada alur kerja sistem pada Docker Swarm dilakukan *pull image* untuk mengunduh *image* Wordpress dan MySQL yang tersedia pada Docker Registry. Setelah itu baru dapat membangun dan menjalankan kontainer Wordpress dan MySQL. Selama kontainer berjalan maka Wordpress akan berjalan dan dapat diakses *user* pada jaringan. *User* dapat melakukan perubahan data pada Wordpress. Semua perubahan data Wordpress yang dilakukan oleh *user* akan disimpan di NFS.

3.4.2 Perancangan Sistem pada NFS

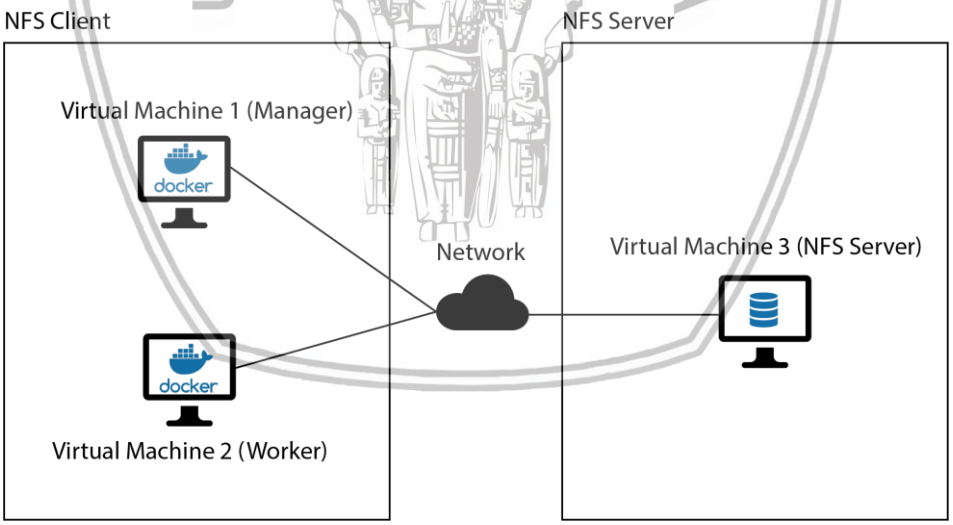
NFS akan menjadi penyimpanan data persisten pada Docker Swarm. NFS menyimpan setiap perubahan data Wordpress yang terjadi pada Docker Swarm. NFS akan selalu melakukan *check* perubahan data Wordpress pad Docker Swarm. Apabila ada perubahan data Wordpress maka NFS akan melakukan sinkroninasi data Wordpress pada Docker Swarm. Gambar 3.3 merupakan flowchart dari alur kerja sistem pada NFS.



Gambar 3.3 Diagram Alir Perancangan Sistem pada NFS

3.4.3 Perancangan Arsitektur Sistem

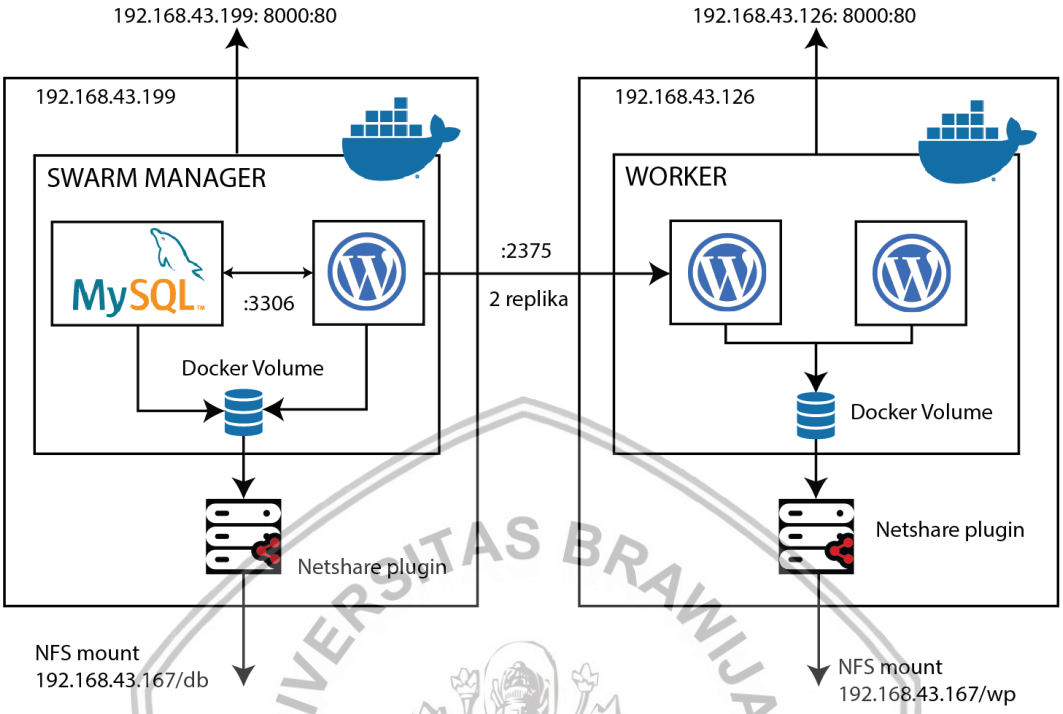
Sistem akan diterapkan pada jaringan Bridged dengan *bandwidth unlimited*. Sistem dibuat dengan arsitektur *client-server*, dimana NFS akan bertidak sebagai *server* terpusat yang menyimpan data dan Docker Swarm akan bertindak sebagai *client* yang menggunakan dan berbagi data melalui NFS Sever. Perancangan arsitektur sistem dapat dilihat pada Gambar 3.4



Gambar 3.4 Perancangan Arsitektur Sistem

Seperti yang terlihat pada Gambar 3.4 merupakan perancangan arsitektur sistem yang akan diimplementasikan untuk penyimpanan data persisten pada Docker Swarm menggunakan NFS. Sistem ini membutuhkan tiga mesin virtual sebagai komponen utama, dua mesin virtual akan bertindak sebagai Docker Swarm dan satu lagi sebagai NFS Server.

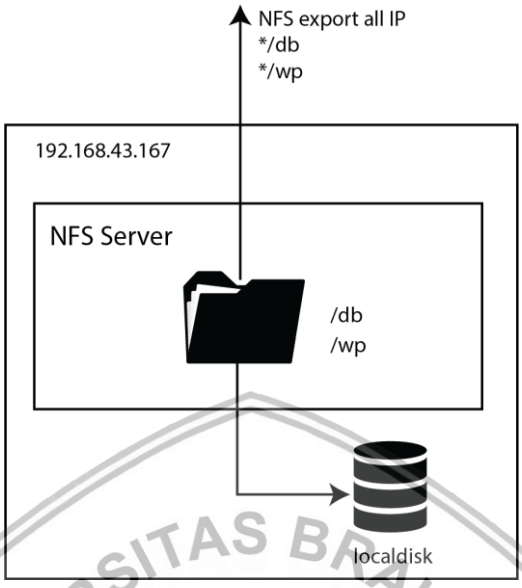
3.4.4 Perancangan Arsitektur Docker Swarm



Gambar 3.5 Perancangan Arsitektur pada Docker Swarm

Gambar 3.5 merupakan detail perancangan arsitektur pada kedua mesin virtual yang berjalan sebagai Docker Swarm. Docker Swarm menjalankan aplikasi Wordpress untuk membangun sebuah layanan web yang dapat diakses *user* untuk melakukan *upload file* dan memerlukan penyimpanan persisten. Wordpress merupakan sebuah aplikasi *open source* yang digunakan untuk manajemen konten blog maupun *website*. Wordpress dibangun dengan bahasa pemrograman PHP dan basis data MySQL. Mesin virtual dengan IP 192.168.43.199 akan menjadi Swarm Manager yang melakukan semua manajemen pada Docker Swarm dan mesin virtual dengan IP 192.168.43.126 akan menjadi Worker yang terhubung dengan Swarm Manager pada port 2357. Keduanya akan membangun Wordpress dan MySQL di dalam kontainer-kontainer. Wordpress akan di-*publish* pada IP mesin virtual dengan port 8000:80. Untuk penyimpanan data kontainer perlu dibuat sebuah Docker Volume yang nantinya akan dihubungkan dengan Docker Volume Plugin Netshare. Melalui Docker Volume Plugin, Docker Volume akan di-*mount* ke direktori yang telah diekspor oleh NFS dengan IP 192.168.43.167.

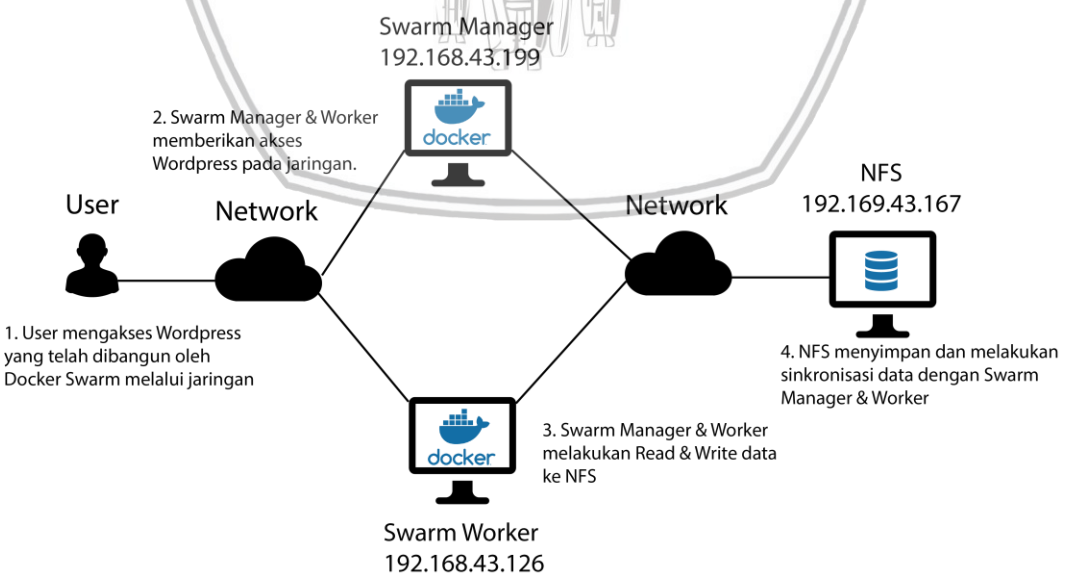
3.4.5 Perancangan Arsitektur NFS



Gambar 3.6 Perancangan Arsitektur pada NFS

Gambar 3.6 merupakan detail perancangan arsitektur penyimpanan data menggunakan NFS Server pada mesin virtual dengan IP 192.168.43.167. NFS server akan membuat direktori /db dan /wp yang akan diekspor ke jaringan sehingga dapat diakses melalui jaringan dan digunakan sebagai penyimpanan. Data disimpan oleh NFS ke penyimpanan lokal.

3.4.6 Perancangan Topologi Sistem



Gambar 3.7 Perancangan Topologi Sistem

Gambar 3.7 merupakan perancangan topologi sistem. Perancangan topologi sistem menunjukan alur kerja dari sistem. *User* dapat mengakses Wordpress

melalui jaringan. Wordpress dibangun oleh Swarm Manager dan Worker dan di-*publish* ke jaringan agar dapat diakses *user*. Ketika *user* melakukan *upload* gambar atau *posting* pada Wordpress maka Swarm Manager dan Worker akan melakukan *write* data ke NFS untuk disimpan. Swarm Manager dan Worker juga dapat *read* data yang telah tersimpan pada NFS. NFS akan selalu melakukan sinkronisasi data pada Swarm Manager dan Worker.

3.5 Implementasi dan Pengujian

Setelah menyelesaikan perancangan sistem, implementasi dilakukan sesuai dengan perancangan sistem yang telah dibuat. Implementasi berupa fase-fase yang terjadi pada implementasi Docker Swarm dalam membangun sistem terdistribusi Wordpress dan MySQL serta fase-fase dalam membangun NFS Server sebagai penyimpanan data persisten pada Docker Swarm.

Pengujian dalam penelitian ini dibagi menjadi dua yaitu pengujian fungsional dan nonfungsional. Pengujian fungsional untuk memastikan fungsi sistem berjalan dengan baik terutama menguji persistensi data NFS. Pengujian nonfungsional dilakukan untuk menguji kinerja kecepatan *write* dan *read* data pada NFS.

3.6 Hasil dan Analisis

Dari data hasil pengujian yang didapatkan, dilakukan analisis untuk mengetahui hasil yang akan digunakan untuk menarik kesimpulan dari penelitian yang dilakukan.

3.7 Penarikan Kesimpulan

Kesimpulan merupakan tahap akhir dan dilakukan setelah sistem selesai dibangun dimana dari hasil pengujian dan analisis data peneliti menarik kesimpulan terkait rumusan masalah.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Sistem

Berdasarkan dari analisis kebutuhan minimum *system requirement* yang telah disebutkan di atas, maka disiapkan beberapa komponen yang diperlukan sebagai berikut:

Tabel 4.1 Spesifikasi laptop

CPU	Intel® Core™ i3-2350M CPU @ 2.30GHz
Memory	8 GB
Disk	640 GB
OS	Windows 10 Pro

Tabel 4.1 merupakan spesifikasi satu unit laptop yang digunakan untuk mengimplementasikan mesin virtual untuk membangun sistem.

Tabel 4.2 Pengaturan Mesin Virtual Docker Swarm

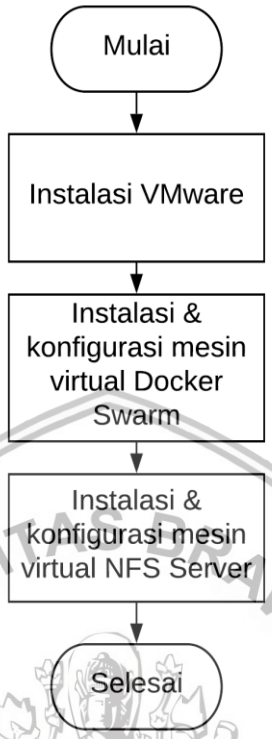
CPU	Dual Core, 2.30GHz
Memory	2 GB
Disk	20 GB
OS	Ubuntu 16.04.3 LTS
Docker version	1.37
NFS version	4

Tabel 4.2 merupakan spesifikasi mesin virtual untuk membangun Docker Swarm. Tabel 4.3 merupakan spesifikasi mesin vitual untuk membangun NFS

Tabel 4.3 Pengaturan Mesin Virtual NFS

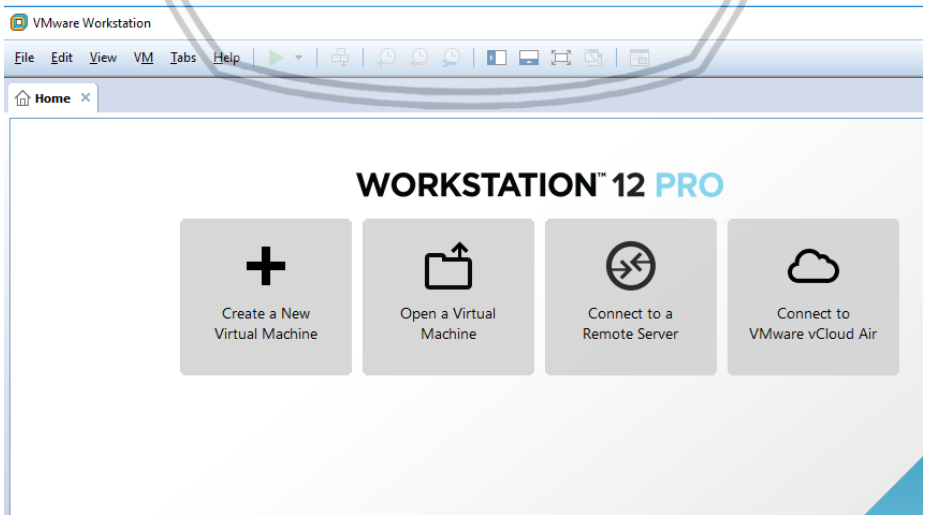
CPU	Dual Core, 2.30GHz
Memory	2 GB
Disk	20 GB
OS	Ubuntu 16.04.3 LTS
NFS version	4

4.2 Implementasi Sistem



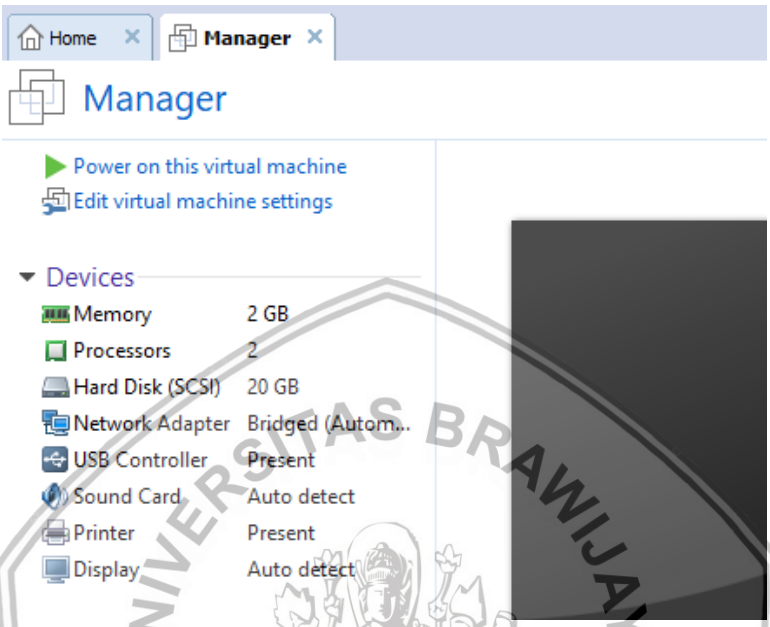
Gambar 4.1 Implementasi Lingkungan Sistem

Implementasi dilakukan dengan cara mengikuti perancangan yang telah dibuat pada bab sebelumnya. Dalam implementasi digunakan sebuah laptop dan perangkat lunak yaitu VMWare. Sebelum mengimplementasikan perancangan Docker Swarm dan NFS terlebih dahulu mempersiapkan lingkungan sistem. Hal pertama yang dilakukan adalah instalasi VMWare pada laptop.



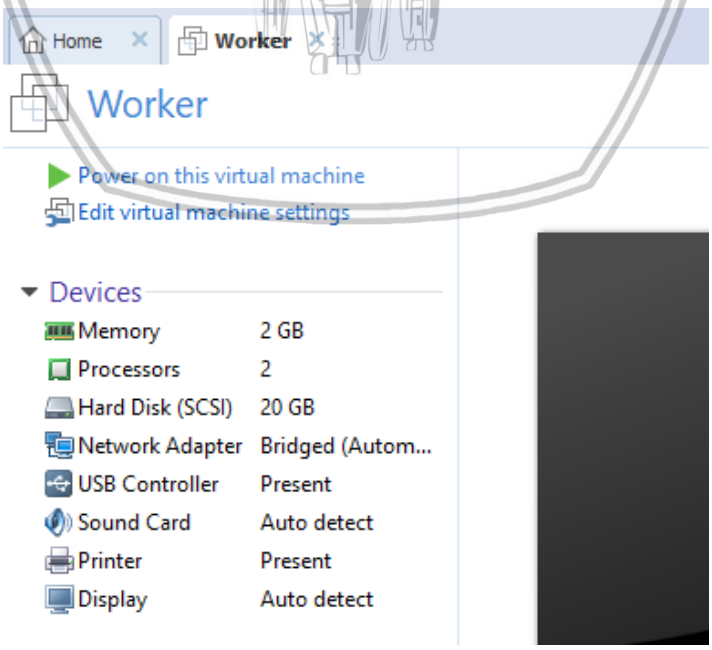
Gambar 4.2 Tampilan VMWare

Gambar 4.2 merupakan tampilan VMWare yang telah diinstal pada laptop. VMWare digunakan untuk mengimplementasikan Docker Swarm dan NFS dalam mesin virtual. Pada VMWare akan disiapkan tiga buah mesin virtual dengan sistem operasi ubuntu 16.04. Spesifikasi sistem pada ketiga mesin virtual akan disesuaikan dengan hasil analisis kebutuhan pada bab sebelumnya.



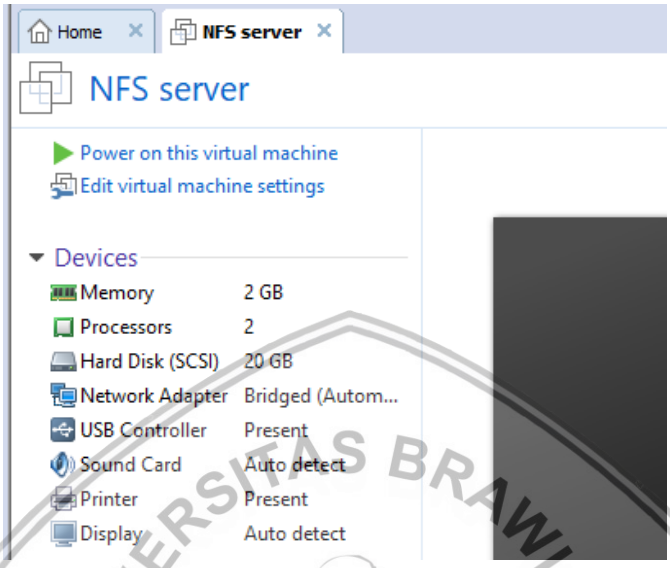
Gambar 4.3 Mesin Virtual Manager

Gambar 4.3 merupakan tampilan mesin virtual Manager yang telah dinstal dan dikonfigurasi. Mesin virtual Manager digunakan sebagai Swarm Manager. Pada mesin virtual Manager diinstal Docker, Docker Compose, Docker Volume Plugin, dan NFS Client.



Gambar 4.4 Mesin Virtual Worker

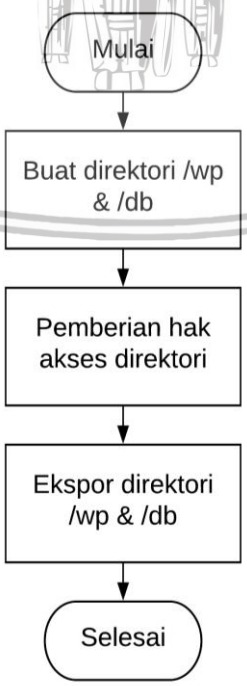
Gambar 4.4 merupakan tampilan mesin virtual Worker yang telah dinstal dan dikonfigurasi. Mesin virtual Worker digunakan sebagai Swarm Worker. Pada mesin virtual Worker juga diinstal Docker, Docker Compose, Docker Volume Plugin, dan NFS Client.



Gambar 4.5 Mesin Virtual NFS Server

Gambar 4.5 merupakan tampilan mesin virtual NFS Server yang telah dinstal dan dikonfigurasi. Mesin virtual NFS Server digunakan sebagai penyimpanan persisten yang akan digunakan oleh Swarm Manager dan Swarm Worker. Pada mesin virtual NFS Server diinstal NFS Server.

4.2.1 NFS



Gambar 4.6 Diagram Alir Implementasi NFS

Gambar 4.6 merupakan diagram alir implementasi NFS. Pada penelitian ini NFS Server diimplementasikan menggunakan mesin virtual dengan IP 192.168.43.167. Untuk mengimplementasikan NFS sebagai penyimpanan data persisten pertama perlu membuat dua direktori baru. Dua direktori yang dibuat yaitu direktori /wp dan direktori /db. Direktori /wp digunakan untuk menyimpan data kontainer Wordpress. Direktori /db digunakan untuk menyimpan data kontainer MySQL.

Kedua direktori /wp dan /db perlu dikonfigurasi agar hak akses dan kepemilikan direktori menjadi *nobody* dan *nogroup*. Fungsi dari *nobody* dan nogroup adalah mengijinkan siapapun mengakses direktori sehingga direktori dapat dengan bebas digunakan bersama.

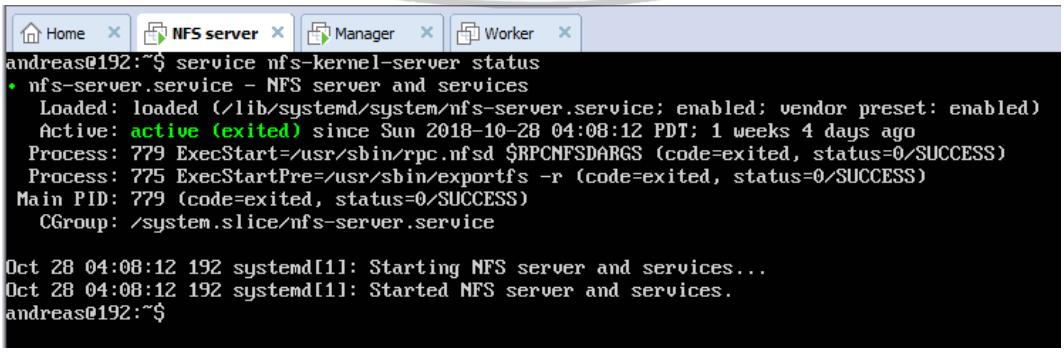
Agar kedua direktori /wp dan /db dapat diekpor ke jaringan perlu dilakukan konfigurasi pada file /etc/exports. Ekspor direktori diperlukan agar direktori tersedia pada jaringan dan dapat diakses oleh *client* melalui jaringan. Sebelum melakukan ekspor direktori /wp dan /db perlu melakukan konfigurasi ekspor seperti pada kode sumber di bawah ini.

Konfigurasi etc/exports

```
1  /wp    *(rw,sync,no_root_squash)
2  /wp    *(rw,sync,no_root_squash)
```

Konfigurasi dengan wildcard "*" digunakan untuk mengijinkan semua IP melakukan *mount* pada direktori, rw (*read-write*) digunakan agar dapat membaca dan menulis data pada direktori, sync digunakan untuk melakukan sinkronisasi data yang tersimpan pada NFS dengan Docker Swarm, no_root_squash digunakan untuk mengijinkan semua *remote client* melakukan akses root.

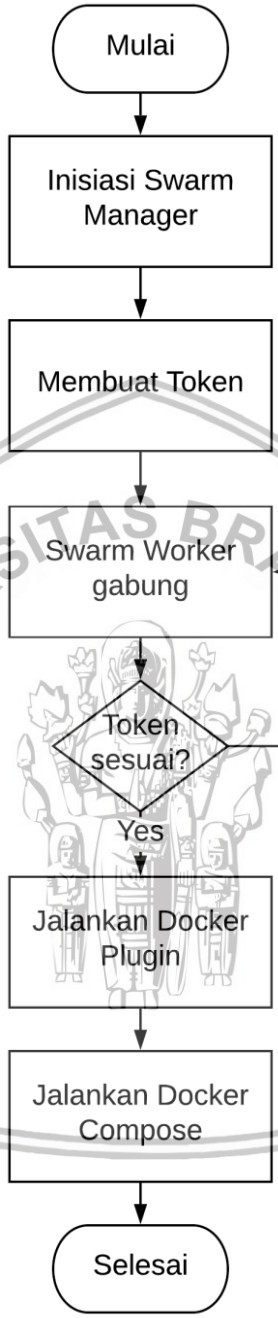
Setelah konfigurasi ekspor selesai selanjutnya adalah melakukan ekpor direktori /wp dan /db. Dengan melakukan ekspor maka direktori akan tersedia untuk di-*mount* oleh Docker Swarm. Ekspor direktori dapat dilakukan dengan menggunakan perintah *exportfs -a*. Dengan memberikan opsi -a pada perintah *exportfs* berarti NFS akan mengekspor semua direktori yang telah dibuat. NFS server dapat dijalankan dengan perintah *service nfs-kernel-server start.*



Gambar 4.7 NFS Berjalan

Gambar 4.7 merupakan tampilan NFS Server yang telah berhasil berjalan pada mesin virtual. NFS Server siap digunakan sebagai penyimpanan Docker Swarm.

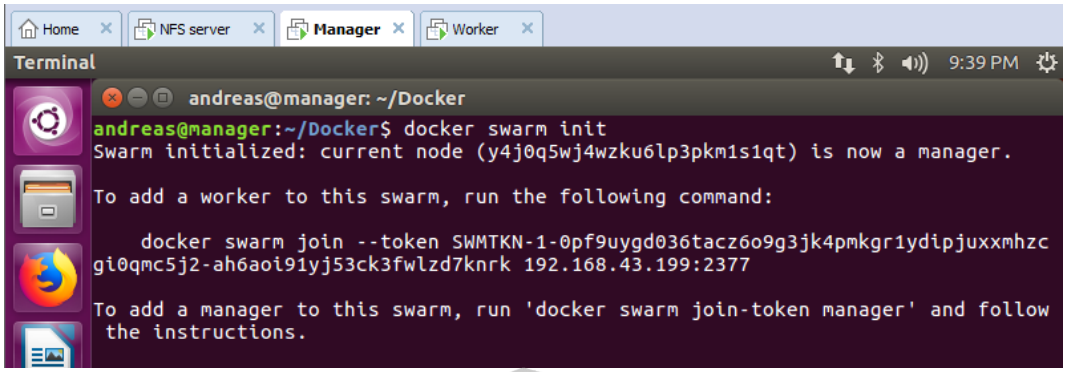
4.2.2 Docker Swarm



Gambar 4.8 Diagram Alir Implementasi Docker Swarm

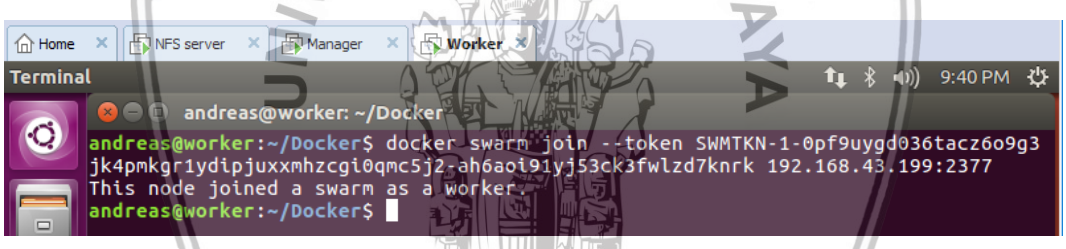
Gambar 4.8 merupakan diagram alir implementasi Docker Swarm. Pada penelitian ini Docker Swarm diimplementasikan menggunakan dua mesin virtual. Mesin virtual dengan IP 192.168.43.199 akan menjadi Swarm Manager. Mesin virtual dengan IP 192.168.43.126 akan berjalan sebagai Swarm Worker. Pada tahap pertama yang dilakukan adalah inisiasi Swarm Manager. Inisiasi Swarm Manager dapat dilakukan dengan perintah *swarm init.* Swarm Manager akan

memberikan sebuah token yang dapat digunakan oleh Swarm Worker untuk bergabung dengan Swarm Manager.



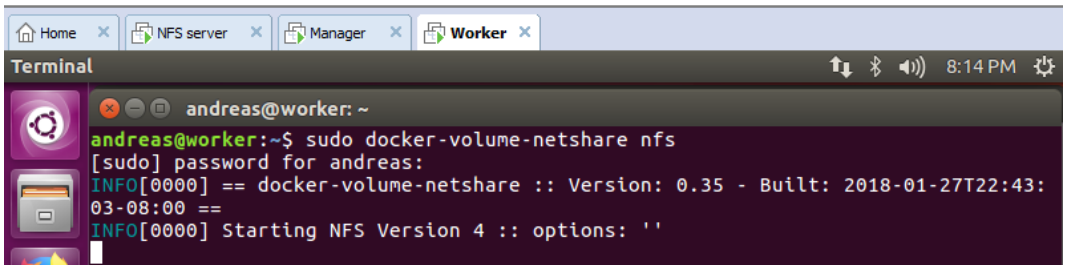
Gambar 4.9 Inisiasi Swarm Manager

Gambar 4.9 merupakan Inisiasi Swarm Manager. Swarm Manager akan memberikan sebuah token yang dapat digunakan oleh Swarm Worker untuk bergabung. Apabila token yang digunakan tidak sama maka Swarm Worker tidak dapat terhubung dengan Swarm Manager.

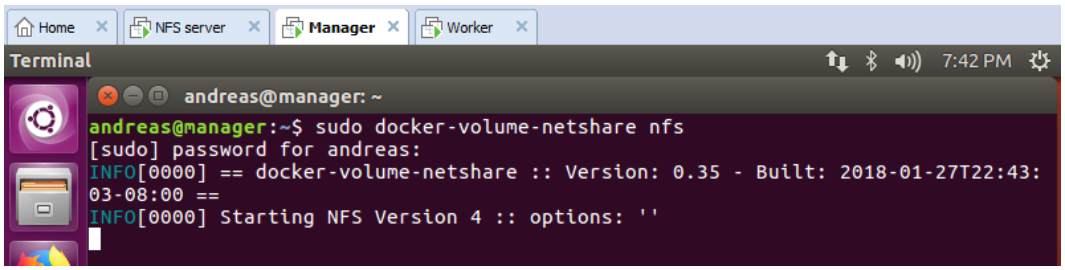


Gambar 4.10 Swarm Worker Bergabung

Gambar 4.10 merupakan tampilan Swarm Worker bergabung dengan Swarm Manager. Setelah Swarm terbentuk, selanjutnya adalah menjalankan Docker Volume Plugin pada Swarm Manager dan Swarm Worker. Docker Volume Plugin digunakan untuk mengakses penyimpanan eksternal NFS. Docker Volume Plugin nantinya akan menghubungkan Docker Swarm dengan NFS secara otomatis ketika Docker Compose dijalankan. Docker Volume Plugin yang berjalan pada Swarm Manager dan Worker dapat dilihat pada Gambar 4.11 dan Gambar 4.12.



Gambar 4.11 Docker Volume Plugin Berjalan pada Swarm Manager



Gambar 4.12 Docker Volume Plugin Berjalan pada Swarm Worker

Untuk membangun dan menjalankan aplikasi-aplikasi pada Docker Swarm hanya dengan sebuah perintah dapat menggunakan Docker Compose. Untuk itu perlu mengkonfigurasi dan mendefinisikan servis yang akan berjalan pada kontainer terlebih dahulu pada file YML.

Konfigurasi docker-compose.yml

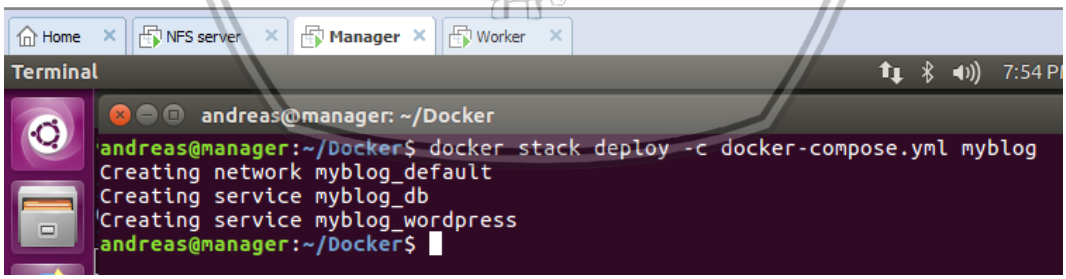
```
1   version: '3.3'
2
3   services:
4     MySQL:
5       image: mysql:5.7
6       volumes:
7         - nfs_db:/var/lib/mysql
8       deploy:
9         placement:
10          constraints: [node.role == manager]
11      environment:
12        MYSQL_ROOT_PASSWORD: wordpress
13        MYSQL_DATABASE: wordpress
14        MYSQL_USER: wordpress
15        MYSQL_PASSWORD: wordpress
16
17    wordpress:
18      depends_on:
19        - db
20      image: wordpress:latest
21      deploy:
22        replicas: 3
23        restart_policy:
24          condition: on-failure
25      volumes:
26        - nfs_wp:/var/www/html
27      ports:
28        - "8000:80"
29      environment:
30        WORDPRESS_DB_HOST: db:3306
31        WORDPRESS_DB_USER: wordpress
32        WORDPRESS_DB_PASSWORD: wordpress
33  volumes:
34    nfs_db:
35      driver: nfs
36      driver_opts:
37        share: 192.168.43.167:/db
```

```
38          nfs_wp:
39            driver: nfs
40            driver_opts:
41              share: 192.168.43.167:/wp
```

Kode sumber konfigurasi docker-compose.yml mendefinisikan dan membangun sebuah servis pada Docker Swarm. Pada Docker Compose, aplikasi yang berjalan disebut *services. Services* yang didefinisikan ada dua yaitu MySQL dan Wordpress. Pada baris 3-15 Didefinisikan sebuah servis MySQL yang dibuat menggunakan i*mage* MySQL versi 5.7. Volume mendefinisikan penyimpanan yang digunakan oleh sebuah servis. *Deploy* merupakan konfigurasi yang berhubungan dengan bagaimana menjalankan sebuah servis. Pada bagian *deploy* pada servis MySQL didefinisikan bahwa *service* MySQL akan berjalan pada Docker Manager. *Environment* mendefinisikan variabel-variabel yang diperlukan oleh *services.* Pada bagian *environment* MySQL didefinisikan variabel yang diperlukan seperti *password* dan *user.*

Pada baris 17-32, mendefinisiakn servis yang kedua yaitu Wordpress. Servis Wordpress dibuat dengan menggunakan *image* Wordpress versi terbaru. *Deploy* pada Wordpress mendefinisikan *replicas* sebanyak tiga dan *restart* ketika kontainer mengalami kegagalan saat dijalankan. *Replicas* digunakan untuk mendefinisikan jumlah spesifik kontainer yang akan dijalankan. *Port* yang digunakan oleh Wordpress adalah 8000:80. *Environment* yang ditambahkan adalah *user* dan *password* untuk mengakses MySQL.

Pada baris 33-41 *volumes* digunakan untuk mendefinisikan Docker Volume sebagai penyimpanan. Pada bagian *volumes* didefinisikan dua Docker Volume dengan nama nfs_db dan nfs_wp, masing-masing volume di-*mount* pada direktori /db dan /wp dengan IP 192.168.43.167.



Gambar 4.13 Menjalankan Docker Compose

Gambar 4.13 merupakan tampilan menjalankan Docker Compose. Untuk menjalankan Docker Compose pada Docker Swarm dapat menggunakan dengan perintah *docker stack deploy.* Opsi -c digunakan untuk *path* ke sebuah *file* Compose.

4.3 Pengujian

Pengujian yang dilakukan dalam penelitian ini dibagi menjadi dua yaitu Pengujian fungsional dan pengujian nonfungsional. Pengujian fungsional

dilakukan untuk menguji apakah semua fungsi pada Docker Swarm, dan NFS dapat berjalan. Pengujian fungsional juga menguji persistensi dari NFS. Pengujian nonfungsional dilakukan untuk menguji kinerja *write* dan *read* data pada NFS.

4.3.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk memastikan semua fungsi sistem berjalan dengan baik. Fungsi-fungsi sistem yang akan diuji meliputi fungsi pada Docker Swarm, dan NFS. Pengujian fungsional pada Docker Swarm dilakukan dengan menjalankan kontainer yang telah didefinisikan sebelumnya pada *file* Docker-Compose.YML. Kontainer yang berjalan dipastikan sesuai dengan perancangan arsitektur Docker Swarm. Swarm Manager menjalankan kontainer MySQL dan kontainer Wordpress. Swarm Worker menjalankan replika kontainer Wordpress.

4.3.1.1 Pengujian Persistensi NFS

Pengujian persistensi NFS dilakukan dengan dua skenario pengujian. Skenario pengujian yang pertama memiliki empat tahap pengujian. Tahap pertama melakukan *upload* dan *posting* pada Wordpress. Tahap kedua melakukan penghapusan semua kontainer pada Docker Swarm. Tahap ketiga membuat dan menjalankan kontainer baru pada Docker Swarm. Tahap terakhir adalah melakukan pengecekan data yang telah di-*upload* dan di-*posting* pada Wordpress dan NFS. Data *upload* dan *posting* pada Wordpress dan NFS dipastikan masih ada atau tidak.

Skenario pengujian yang kedua memiliki tiga tahap pengujian. Tahap pertama adalah melakukan *restart* pada semua mesin virtual. Tahap kedua membuat dan menjalankan kontainer baru pada Docker Swarm. Tahap ketiga melakukan pengecekan di-*upload* dan di-*posting* pada Wordpress dan NFS. Data *upload* dan *posting* pada Wordpress dan NFS dipastikan masih ada atau tidak.

4.3.2 Pengujian Non Fungsional

Pengujian Nonfungsional dilakukan untuk mengetahui kinerja kecepatan *write* dan *read* data pada NFS. Untuk menguji kinerja NFS dapat dilakukan dengan menggunakan perintah Nfsiostat. Nfsiostat adalah sebuah perintah pada Linux untuk menampilkan statistik mengenai operasi-operasi *read* dan *write* pada NFS. Nfsiostat dapat ditampilkan dalam satuan waktu detik. Tabel 4.4 adalah output yang akan ditampilkan nfsiostat:

Tabel 4.4 Output nfsiostat

ops/s	Jumlah operasi per detik
KB/s	Jumlah KB write/read per detik
KB/op	Jumlah KB write/read per operasi
avg RTT (ms)	Waktu yang diperlukan untuk mengirim *request* hingga menerima *reply*

4.3.2.1 Pengujian Kinerja Kecepatan *Write* pada NFS

Untuk mengetahui kinerja kecepatan *write* pada NFS dilakukan dengan meng-*upload file* gambar pada Wordpress dengan ukuran 285 MB. *File* gambar di-*upload* untuk melihat kinerja kecepatan *write* pada NFS dengan menggunakan perintah Nfsiostat. Nfsiostat akan menampilkan kinerja kecepatan *write* pada NFS yang ditampilkan setiap interval satu detik. Masing-masing skenario pengujian dilakukan sebanyak lima kali agar hasil yang didapat lebih akurat.

4.3.2.2 Pengujian Kinerja Kecepatan *Read* pada NFS

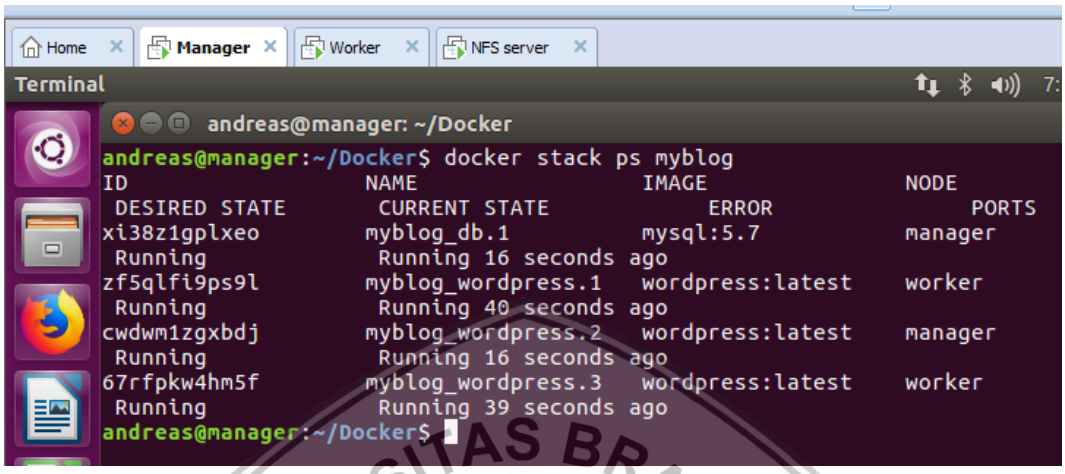
Skenario pengujian kinerja kecepatan *read* data pada NFS dilakukan dengan melakukan penghapusan semua kontainer. Kemudian membuat serta menjalankan kontainer baru. Pada saat membuat dan menjalankan kontainer yang baru dilakukan *monitoring* kinerja *read* data pada NFS dengan menggunakan Nfsiostat. Skenario pengujian dilakukan sebanyak lima kali untuk mendapat data yang akurat.

4.3.2.3 Pengujian Kinerja Kecepatan *Read* dan *Write* pada Mesin Virtual

Pengujian ini dilakukan dilakukan untuk mengetahui perbandingan kinerja kecepatan *read* dan *write* antara mesin virtual dengan NFS. Dalam melakukan pengujian kecepatan *write* data pada mesin virtual digunakan perintah dd. Perintah dd merupakan sebuah perintah pada linux yang dapat digunakan untuk melakukan tes kecepataan *read-write* data pada Disk mesin virtual. Pengujian read & write dilakukan sebanyak lima kali untuk mendapatkan hasil yang lebih akurat.

BAB 5 HASIL DAN ANALISIS

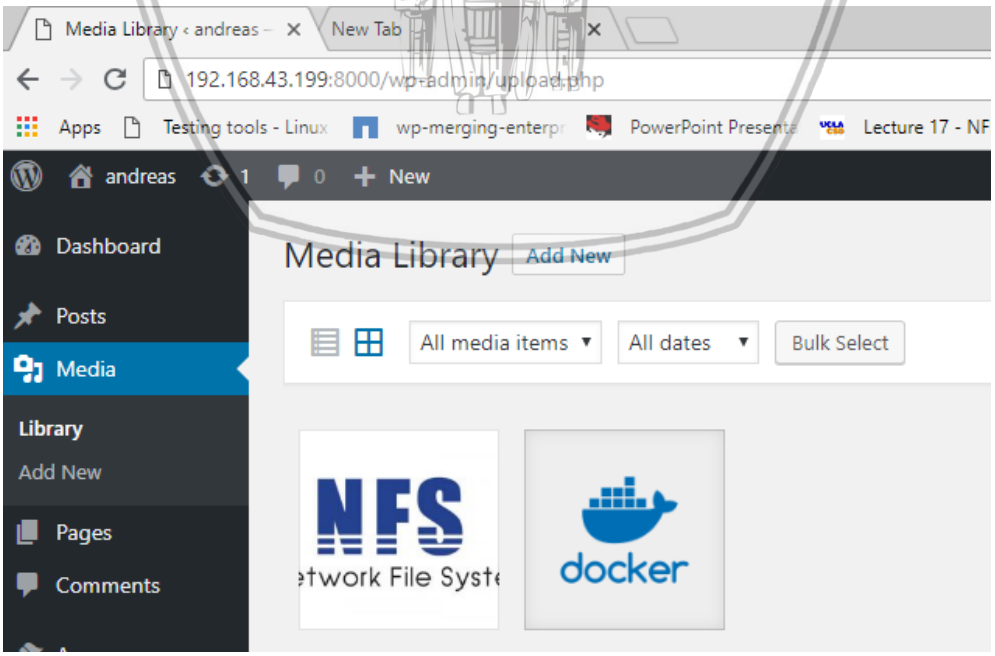
5.1 Hasil Pengujian Fungsional



Gambar 5.1 Kontainer yang Berjalan

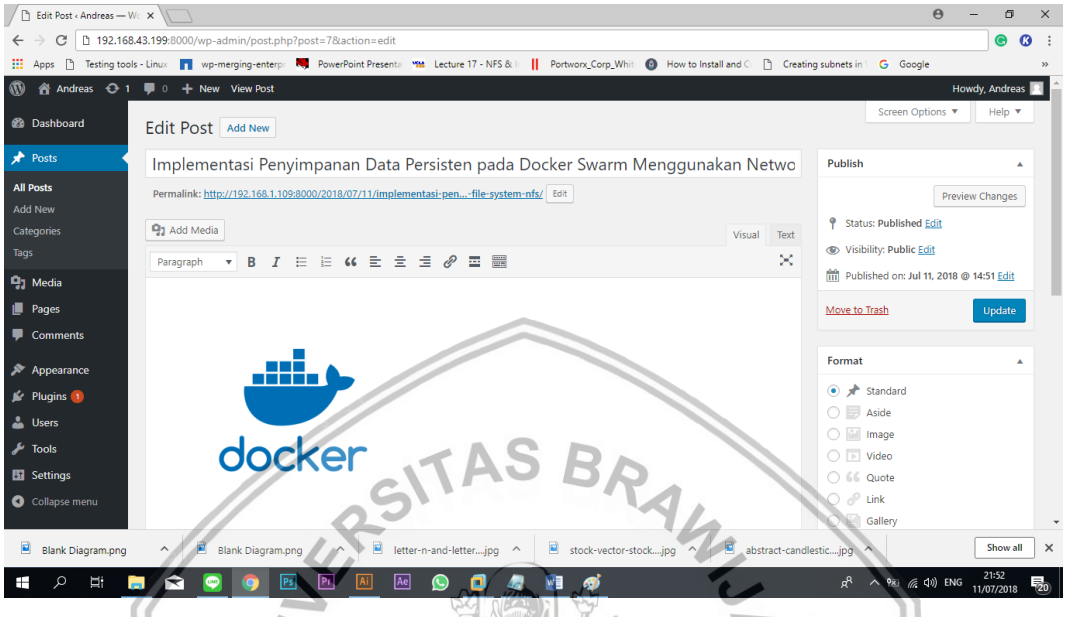
Gambar 5.1 memperlihatkan bahwa Docker Swarm sedang menjalankan empat buah kontainer. Empat buah kontainer terdiri dari tiga kontainer Wordpress dan satu kontainer MySQL. Kontainer MySQL dan satu kontainer Wordpress berjalan pada Swarm Manager. Dua kontainer Wordpress berjalan pada Swarm Worker.

5.1.1 Hasil Pengujian Persistensi NFS



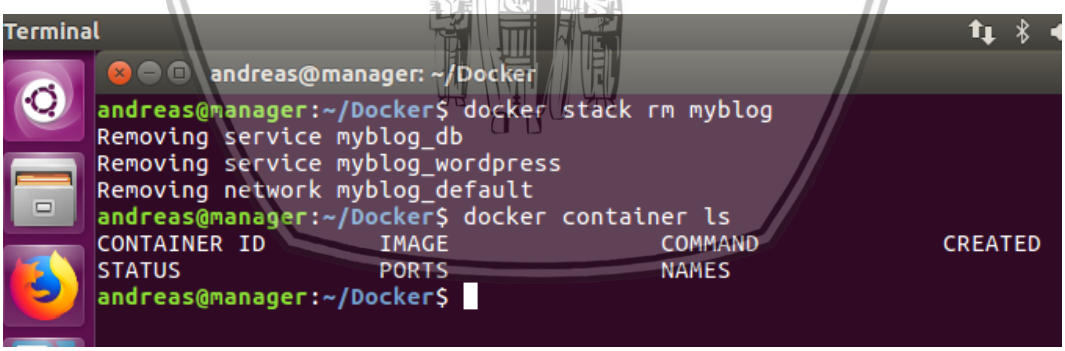
Gambar 5.2 Tampilan *Upload* Gambar pada Wordpress

Gambar 5.2 memperlihatkan gambar-gambar yang telah berhasil di-*upload* pada Wordpress. Gambar yang di-*upload* pada Wordpress akan secara otomatis tersimpan pada NFS Server.



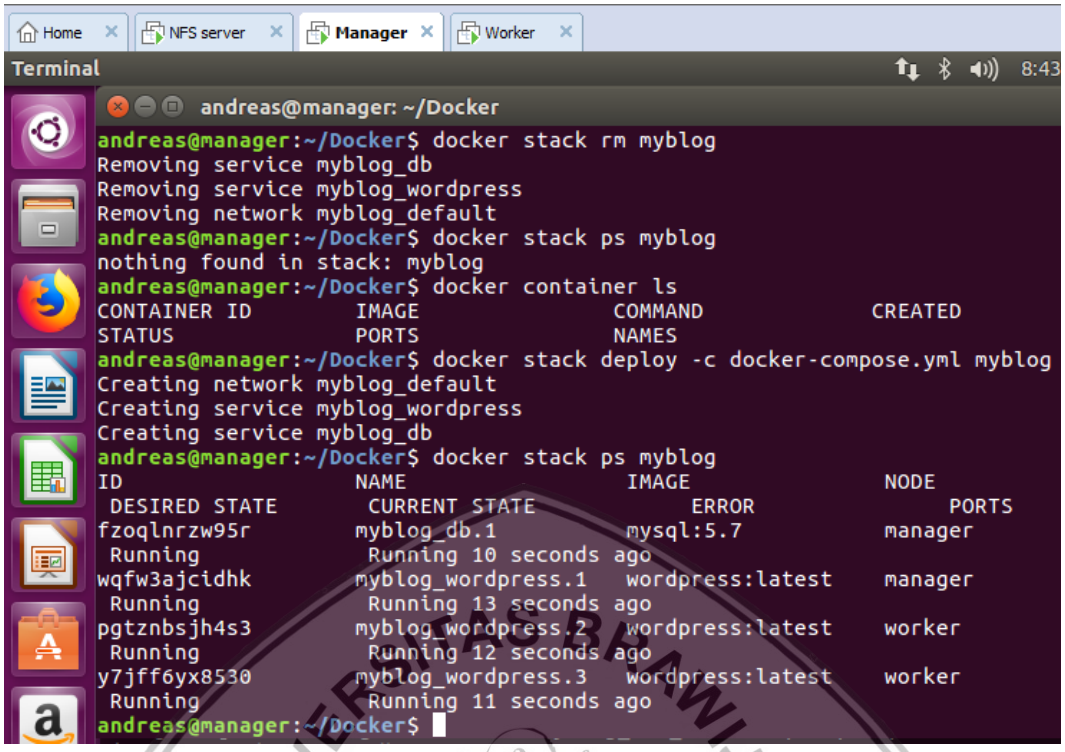
Gambar 5.3 Tampilan *Posting* pada Wordpress

Gambar 5.3 memperlihatkan tampilan *posting* pada Wordpress. Setelah melakukan *posting* pada Wordpress data akan tersimpan pada NFS.



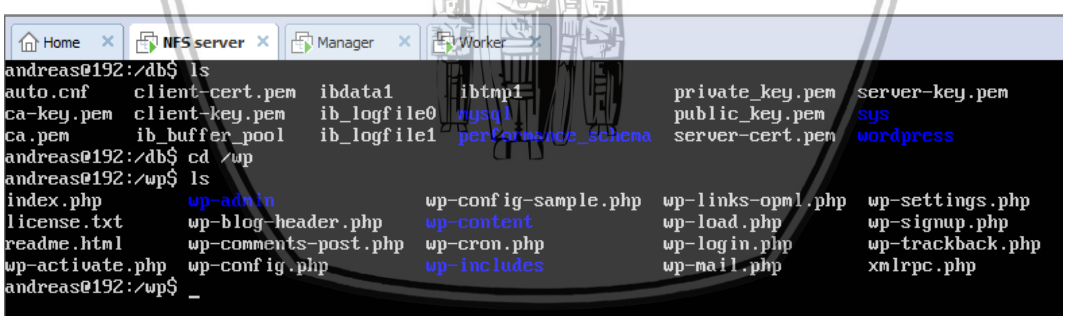
Gambar 5.4 Tampilan Kontainer dihapus

Gambar 5.4 memperlihatkan penghapusan semua kontainer yang berjalan pada Docker Swarm.



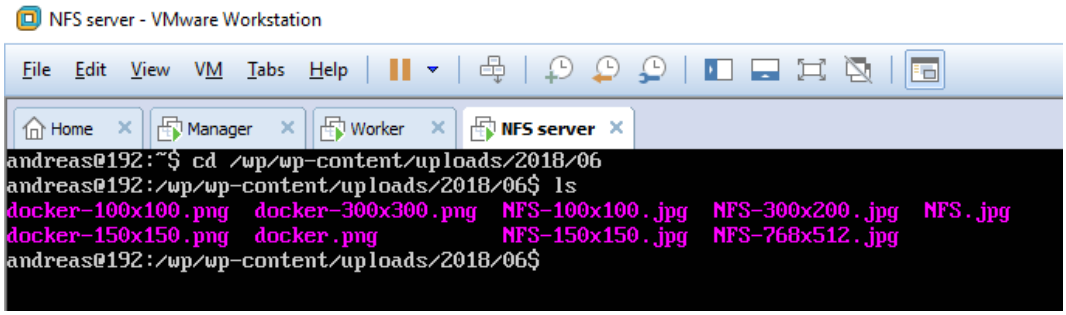
Gambar 5.5 Tampilan Kontainer Membuat dan Menjalankan Kontainer Baru

Gambar 5.5 memperlihatkan Docker Swarm membuat dan menjalankan kontainer baru setelah kontainer dihapus. Kemudian dilakukan pengecekan data pada NFS untuk memastikan data masih ada atau tidak.



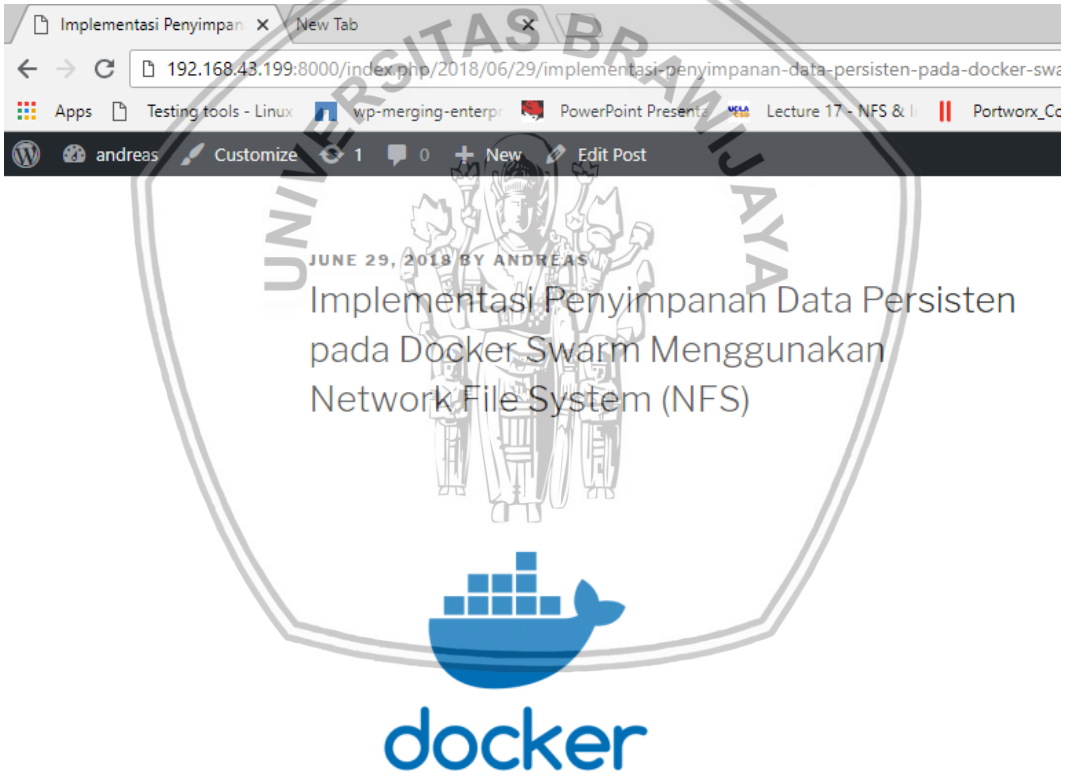
Gambar 5.6 Data Kontainer Wordpress dan MySQL yang Tersimpan pada NFS

Gambar 5.6 memperlihatkan bahwa data Wordpress dan MySQL tersimpan pada NFS. Segera setelah Docker Swarm membangun Wordpress dan MySQL, data pada kontainer Wordpress dan MySQL tersimpan secara otomatis pada NFS. Data Wordpress tersimpan pada direktori /wp dan data MySQL tersimpan pada direktori /db.



Gambar 5.7 Tampilan Data *Upload* yang Tersimpan pada NFS

Gambar 5.7 memperlihatkan bahwa gambar yang telah di-*upload* pada Wordpress tersimpan pada NFS Server. Data *upload* tersimpan pada direktori /wp/wp-content/uploads/2018/06. Secara *default* Wordpress menyimpan gambar yang di-*upload* dengan tiga ukuran berbeda dan ukuran asli seperti yang pada gambar.



Gambar 5.8 Tampilan *Posting* pada Wordpress

Setelah melakukan pengecekan data pada NFS selanjutnya adalah pengecekan data pada Wordpress. Gambar 5.8 memperlihatkan tampilan *posting* pada Wordpress. Tampilan *posting* pada Gambar 5.8 adalah setelah kontainer dihapus, dan semua mesin virtual di-*restart.* Didapatkan bahwa data pada *upload* gambar dan *posting* pada Wordpress masih persisten.

Tabel 5.1 Pengujian Fungsional

NO.	Pengujian	Hasil
1.	Swarm Worker dapat terhubung dengan Swarm Manager	Valid
2.	Docker Swarm dapat menjalankan kontainer	Valid
3.	Docker Swarm dapat melakukan *read* dan *write* data pada NFS	Valid
4.	NFS dapat menyimpan data kontainer	Valid
5.	NFS dapat melakukan sinkronisasi data pada Docker Swarm	Valid

Tabel 5.1 merupakan daftar tabel hasil pengujian fungsional yang telah dilakukan di atas.

5.2 Hasil Pengujian NonFungsional

5.2.1 Hasil Pengujian Kinerja Kecepatan *Write* NFS

Tabel 5.2 Output *Write* NFS

output	Interval (detik)									
	1	2	3	4	5	6	7	8	9	10
ops/s	136	158	182	183	172	171	158	155	144	129
KB/s	31217	31553	30475	30309	28057	28730	30159	28340	28394	27218
KB/op	230	204	167	166	165	168	192	194	204	219
avg RTT (ms)	238	245	257	265	275	442	254	299	246	262

Pengujian kinerja kecepatan *write* NFS dilakukan dengan menggunakan *file* berukuran 285 MB. *File* di-*upload* sebanyak lima kali dan hasil pengujian di-*monitoring* menggunakan nfsiostat. Hasil pengujian kemudian dirata-ratakan dan ditampilkan pada tabel 5.2.

5.2.2 Hasil Pengujian Kinerja Kecepatan *Read* NFS

Tabel 5.3 Output *Read* NFS

output	interval (detik)									
	1	2	3	4	5	6	7	8	9	10
ops/s	300	306	358	352	346	348	356	366	356	341
KB/s	66410	65648	64638	63159	63106	63210	62422	60618	60318	59866
KB/op	232	205	180	181	182	181	178	166	169	179
avg RTT (ms)	68	247	245	281	211	265	224	265	254	247

Tabel 5.3 merupakan hasil pengujian kinerja kecepatan *read* NFS. Ketika kontainer baru dibuat dan dijalankan kinerja kecepatan *read* di-*monitoring* menggunakan nfsiostat.

5.2.3 Hasil Pengujian Kinerja Kecepatan *Read* dan *Write* Mesin Virtual

Tabel 5.4 Output *read* dan *write* mesin virtual

Output	Percobaan ke-									
	1	2	3	4	5	6	7	8	9	10
Read MB/s	73.3	73.8	70.7	73.9	72.6	72.7	72.9	71.2	71.4	70.9
Write MB/s	36	36.6	36.9	37	35.6	36	35.8	34.8	35.3	32.2

Tabel 5.4 merupakan hasil pengujian kinerja kecepat *read-write* pada mesin virtual. Kinerja kecepatan *read-write* pada mesin virtual di-*monitoring* menggunakan nfsiostat. Hasil pengujian kecepatan *read-write* yang dilakukan sebanyak sepuluh kali.

5.3 Analisis Pengujian Fungsional

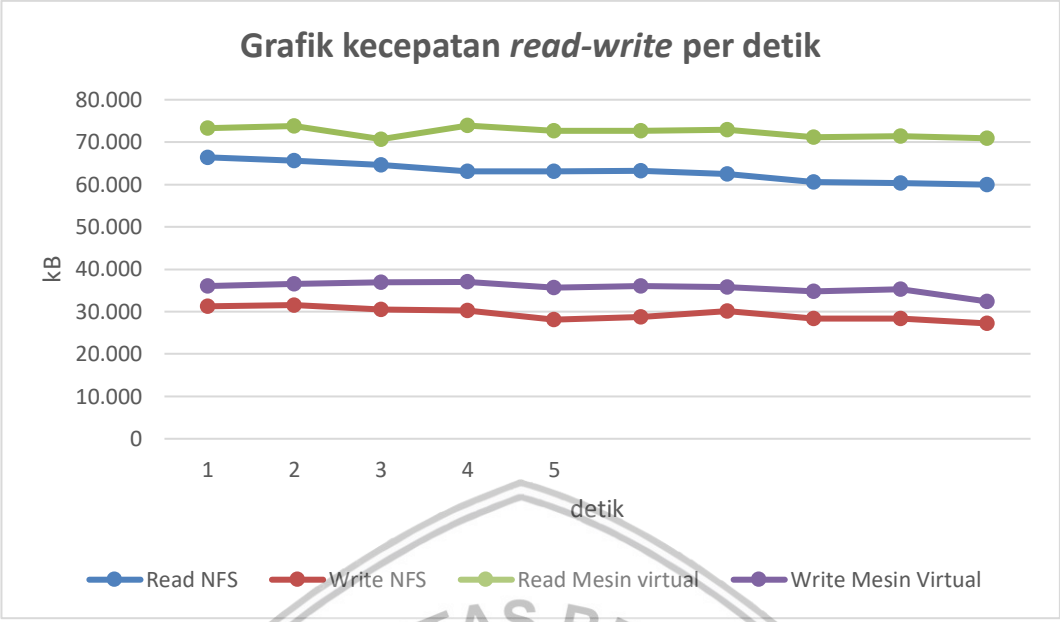
5.3.1 Analisis Persistensi NFS

Pada pengujian persistensi NFS didapat hasil bahwa setelah semua kontainer dihapus maupun mesin virtual di-*restart*, data pada Wordpress dan NFS masih tetap persisten. Ketika user melakukan *upload* gambar dan *posting* pada Wordpress, NFS langsung melakukan sinkronisasi data dengan memanggil prosedur *write* pada kontainer dengan Remote Procedure Call (RPC), sehingga kontainer secara otomatis langsung melakukan *write* dan menyimpan data ke NFS. Oleh karena itu, sekalipun semua kontainer dihapus, data Wordpress telah tersimpan pada NFS. Ketika membuat dan menjalan kontainer baru maka NFS kembali melakukan sinkronisasi data pada kontainer baru yang terhubung. NFS akan memanggil prosedur *read* pada kontainer dengan RPC, sehingga kontainer secara otomatis langsung melakukan *read* dan mengambil data dari NFS.

5.4 Analisis Pengujian NonFungsional

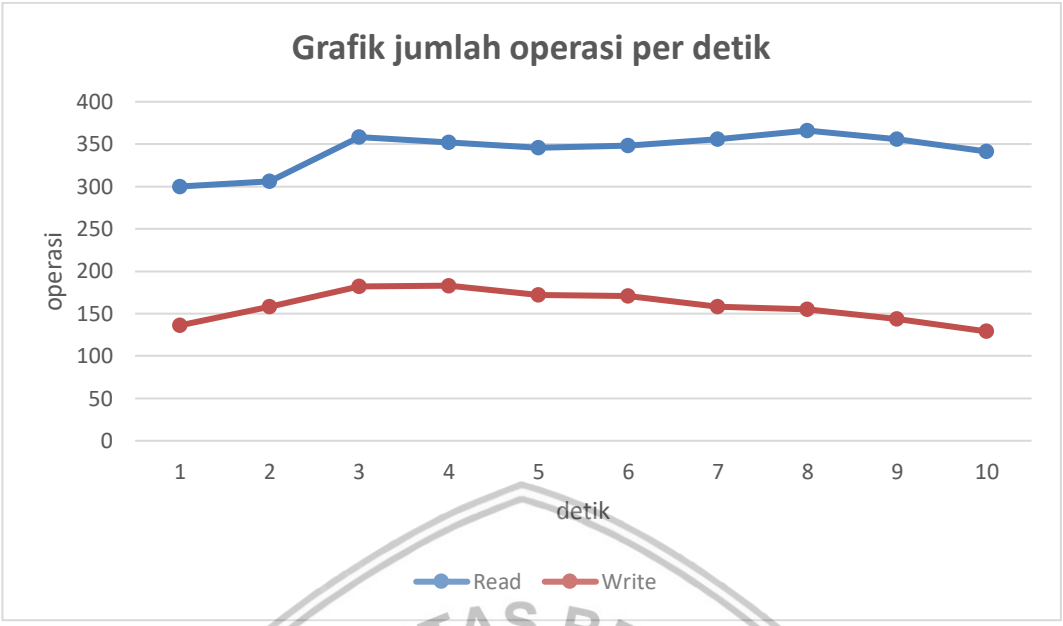
5.4.1 Analisis Perbandingan Kinerja *Read* dan *Write* NFS dengan Mesin Virtual

Analisis kinerja kecepatan *read* dan *write* pada NFS dilakukan dengan menggunakan data yang diperoleh dari pengujian nonfungsional sebelumnya. Data hasil pengujian kinerja kecepatan *read* dan *write* pada NFS diubah menjadi grafik untuk memudahkan analisis. Berikut grafik pengujian kinerja kecepatan *read-write* data pada NFS dan mesin virtual dapat dilihat pada Gambar 5.9.



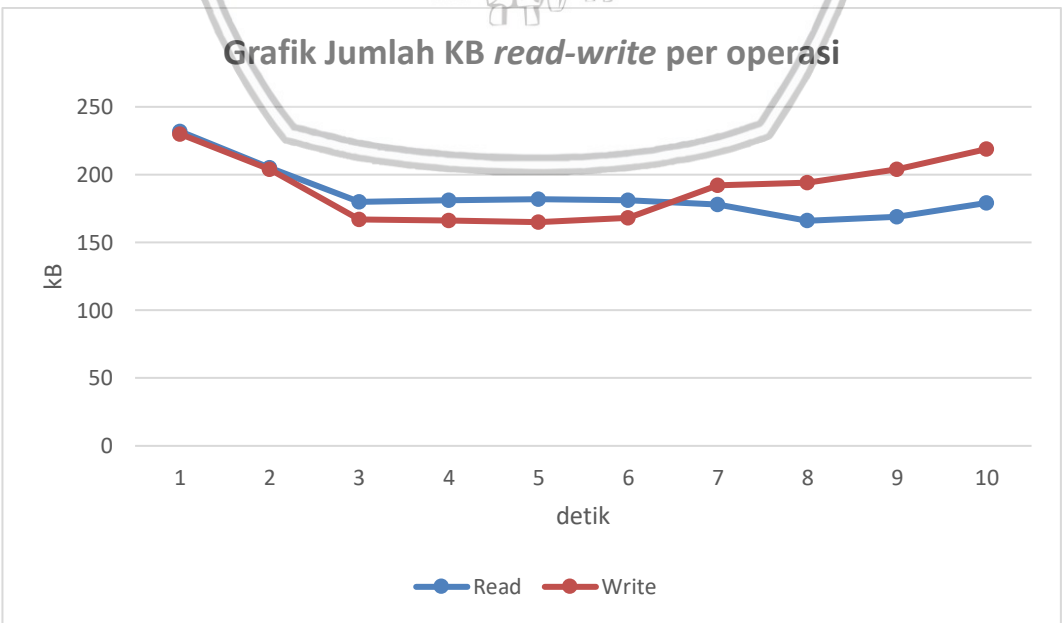
Gambar 5.9 Grafik Kecepatan *Read-Write* per Detik

Gambar 5.9 merupakan grafik kecepatan *read-write* per detik pada NFS dan mesin virtual. Kecepatan *write* rata-rata NFS adalah 30.168 KB sedangkan kecepatan read rata-rata NFS adalah 63.939 KB. Kecepatan *write* rata-rata mesin virtual adalah 35.620 KB sedangkan kecepatan read rata-rata NFS adalah 72.340 KB. Kecepatan *read-write* NFS yang diperoleh cukup cepat karena mendekati kecepatan *read-write* mesin virtual. Pada keduanya hanya terdapat perbedaan kecepatan *write* sebesar 5 MB/s dan kecepatan read sebesar 9 MB/s. Hal ini sebabkan karena pada pengujian kecepatan *read-write* pada mesin virtual, data secara langsung melakukan *read* dan *write* ke *disk*. Pada pengujian kecepatan *read-write* pada NFS, data dikirim melewati jaringan sehingga kecepatan *write* NFS akan dipengaruhi oleh banyak faktor diantanya yaitu *bandwidth* jaringan, besar blok data, jumlah blok data yang dikirim per detik, dan RTT.



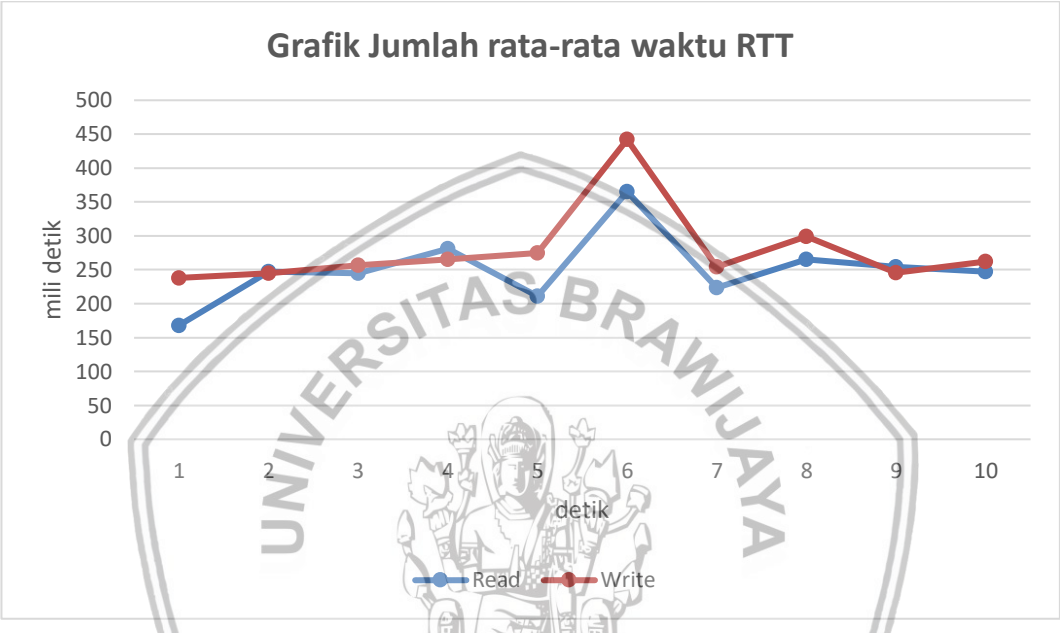
Gambar 5.10 Jumlah Operasi per Detik

Gambar 5.10 merupakan grafik jumlah operasi atau jumlah blok data yang dikirim tiap detiknya. Jumlah blok data yang dikirim per detik paling rendah adalah 122 blok data dan paling tinggi adalah 183 blok data. Pada detik awal terlihat bahwa jumlah blok data yang dikirim semakin banyak, hal ini disebabkan karena CPU akan memaksimalkan pengiriman blok data yang dapat di-*read* dan di-*write* oleh *disk*. Mendekati pertengahan jumlah blok data yang dikirim mengalami penurunan, hal ini disebabkan karena kerja *disk* semakin sibuk. Jumlah blok data yang dikerjakan akan mempengaruhi kecepatan *read* dan *write* NFS, semakin banyak jumlah blok data yang dikerjakan maka waktu yang diperlukan untuk menyimpan sebuah file akan semakin cepat.



Gambar 5.11 Jumlah KB *Read-Write* per Operasi

Gambar 5.11 menunjukan grafik jumlah KB *read-write* per operasi yang merupakan ukuran blok data yang dikirim. Grafik menunjukan besar ukuran blok data yang paling besar yaitu 250 KB dan paling kecil 165 KB. Besar ukuran blok data pada NFS adalah 256 KB sehingga ukuran blok yang dikirim tidak akan lebih besar dari 256 KB. Pada grafik diketahui bahwa besar ukuran blok data semakin menurun kemudian mendekati stabil. Hal ini dikarenakan NFS sedang melakukan penyesuaian besar ukuran blok data yang akan dikirim. Besar blok data akan mempengaruhi kecepatan *read-write* data.



Gambar 5.12 Jumlah Waktu Rata-Rata RTT

Pada gambar 5.12 menunjukan grafik jumlah rata-rata waktu RTT per mili detik. NFSv4 menggunakan jaringan TCP/IP untuk pengiriman data. Pada jaringan TCP/IP terdapat RTT. RTT disini merupakan jumlah waktu yang diperlukan oleh Docker Swarm untuk mengirimkan paket ke NFS hingga menerima kembali respon pengiriman dari NFS. Grafik menunjukan bahwa jumlah waktu RTT semakin tinggi karena ketika Server dibanjiri dengan *request* maka waktu respon Server akan semakin besar. Selain itu jumlah rata-rata RTT juga dipengaruhi oleh kepadatan jaringan, semakin padat jaringan maka akan semakin besar waktu rata-rata RTT.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan hasil dari implementasi, pengujian, dan analisi dari penyimpanan data persisten pada Docker Swarm menggunakan NFS, dapat disimpulkan bahwa:

1. Perancangan arsitektur penyimpanan data persisten pada Docker Swarm menggunakan NFS dirancang menggunakan arsitektur *client-server*. Docker Swarm berperan sebagai *client* dan NFS berperan sebagai *server*. Docker Swarm dirancang untuk menjalankan aplikasi-aplikasi sistem terdistribusi yang memerlukan penyimpanan persisten. Dalam penelitian ini aplikasi-aplikasi yang digunakan yaitu Wordpress dan MySQL. NFS dirancang sebagai penyimpanan eksternal untuk menyimpanan data Wordpress dan MySQL. Untuk menghubungkan Docker Swarm dengan NFS diperlukan juga Docker Volume Plugin.
2. NFS mampu menyediakan penyimpanan data persisten pada Docker Swarm sekalipun kontainer dihapus dan mesin di-*restart*. NFS bertanggungjawab untuk melakukan sinkronisasi data pada Docker Swarm. NFS dapat memanggil prosedur *read-write* menggunakan RPC. Apabila data pada Docker Swarm terjadi perubahan maka NFS melakukan sinkronisasi data. NFS akan memanggil prosedur *write* pada Docker Swarm. *Write* dilakukan untuk menyimpan perubahan data pada NFS. Ketika Kontainer terhapus kemudian dibuat dan dijalankan kembali, maka NFS akan melakukan sinkronisasi data. Sinkronisasi data dilakukan dengan memanggil prosedur *read* pada Docker Swarm. *Read* dilakukan untuk mengambil dan menggunakan data yang telah tersimpan pada NFS.
3. Berdasarkan hasil pengujian kinerja kecepatan *read-write* NFS, dibandingkan kinerja kecepatan *read-write* data antara mesin virtual dengan NFS. Kecepatan *write* rata-rata NFS adalah 30.168 KB sedangkan kecepatan *read* rata-rata NFS adalah 63.939 KB. Kecepatan *write* rata-rata mesin virtual adalah 35.620 KB sedangkan kecepatan read rata-rata NFS adalah 72.340 KB. Kecepatan *read-write* NFS yang diperoleh cukup cepat karena mendekati kecepatan *read-write* mesin virtual. Pada keduanya hanya terdapat perbedaan kecepatan *write* sebesar 5 MB/s dan kecepatan *read* sebesar 9 MB/s. Hal ini dikarenakan *write* data pada NFS harus melalui jaringan TCP/IP sehingga kecepatan akan dipengaruhi oleh *bandwidth* jaringan, ukuran blok data, jumlah blok data yang dikirim per detik, dan RTT.

6.2 Saran

Saran yang dapat disampaikan penulis untuk pengembangan penyimpanan data persisten pada Docker Swarm menggunkan NFS adalah:

1. Perlu dilakukan penelitian lebih lanjut untuk membangun sistem dengan skala yang lebih besar pada Docker Swarm untuk menguji kinerja NFS.
2. Perlu dilakukan pengujian lebih lanjut terhadap *read* data pada NFS .

DAFTAR PUSTAKA

Canonical Ltd, 2004. Network File System (NFS). [Online] Tersedia di: <https://help.ubuntu.com/lts/serverguide/network-file-system.html> [diaskes 10 April 2018]

Docker, Inc., 2008. Overview of Docker Compose. [Online] Tersedia di: <https://docs.docker.com/compose/overview/> [diakses 10 April 2018]

Docker, Inc., 2008. Supported drivers. [Online] Tersedia di: <https://docs.docker.com/storage/volumes/> [diakses 1 Februari 2018]

Docker, Inc., 2008. Docker Overview. [Online] Tersedia di: <https://docs.docker.com/engine/docker-overview/> [diakses 1 Februari 2018]

Docker, Inc., 2008. What is a Container. [Online] Tersedia di: <https://www.docker.com/what-container> [diakses 16 Mei 2018]

Dua, R., Raja, R., & Kakadia, D., 2014. *Virtualization vs Containerization to support PaaS*. IEEE International Conference on Cloud Engineering, pp. 610-614.

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

Gerber, A., 2015. *The State of Containers and the Docker Ecosystem.* United States of America: O'Reilly Media, Inc.

Giles, E.R, 2016. *Container-Based Virtualization for Byte-Addressable NVM Data Storage*. IEEE International Conference on Big Data (Big Data), pp. 2754-2763.

Guo, S., Yang, W., & Wang, G., 2013. *NFS Protocol Performance Analysis and Improvement for Mobile Transparent Computing.* IEEE International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing, pp. 1879-1883.

Huang, C., Lee, C., 2017. *Enhancing the Availability of Docker Swarm Using Checkpoint-and-Restore.* 2017 14th International Symposium on Pervasive Systems, Algorithms and Networks & 2017 11th International Conference on Frontier of Computer Science and Technology & 2017 Third International Symposium of Creative Computing, pp. 357-362.

Jones, M., 2010. Network File Systems and Linux. [Online] Tersedia di: <https://www.ibm.com/developerworks/library/l-network-filesystems/index.html> [diakses 20 April 2018]

Kulkarni, O., Gawali, D., Bagul, S., & Meshram, B., 2010. *Study of Network File System(NFS) And Its Variations.* International Journal of Engineering Research and Applications (IJERA). Vol. 1, Issue 3, pp.721-729.

Lee, K., Kim, H., Kim, B., & Yoo, C., 2017. *Analysis on network performance of container virtualization on IoT devices.* Information and Communication Technology Convergence, pp. 35-37.

Naik, N., 2016. *Building A Virtual System of Systems Using Docker Swarm in Multiple Clouds.*

Nguyen, N., Bein, D., 2017. *Distributed MPI Cluster with Docker Swarm Mode*.

Turnbull, J., 2014. *The Docker Book.* [e-book] Tersedia di: Google Books <https://books.google.co.id/> [diakses 1 Mei 2018]