OPTIMASI *SERVER LIVE STREAMING* MENGGUNAKAN *MICROSERVICES* DAN *LOAD BALANCER*

SKRIPSI

oleh:
MUHAMMAD FAHMI KURNIAWAN
NIM. 14650074

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2020

OPTIMASI *SERVER LIVE STREAMING* MENGGUNAKAN *MICROSERVICES* DAN *LOAD BALANCER*

SKRIPSI

Diajukan Kepada:
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

oleh:
MUHAMMAD FAHMI KURNIAWAN
NIM. 14650074

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2020

LEMBAR PERSETUJUAN

OPTIMASI *SERVER LIVE STREAMING* MENGGUNAKAN *MICROSERVICES* DAN *LOAD BALANCER*

SKRIPSI

oleh:
MUHAMMAD FAHMI KURNIAWAN
NIM. 14650074

Telah Diperiksa dan Disetujui untuk Diuji
Tanggal : Juni 2020

Dosen Pembimbing I	Dosen Pembimbing 2
Dr. Ir. M. Amin Hariyadi, M.T	Ajib Hanani, M.T
NIP. 19670118 200501 1 001	NIDT. 19840731 20160801 1 076

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

LEMBAR PENGESAHAN

OPTIMASI *SERVER LIVE STREAMING* MENGGUNAKAN *MICROSERVICES* DAN *LOAD BALANCER*

SKRIPSI

oleh:
MUHAMMAD FAHMI KURNIAWAN
NIM. 14650074

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal: Juni 2020

Susunan Dewan Penguji			**Tanda Tangan**
Penguji Utama	:	Fajar Rohman Hariri, M.Kom NIP. 19890515 201801 1 001	(_____)
Ketua Penguji	:	Irwan Budi Santoso, M.Kom NIP. 19770103 201101 1 004	(_____)
Sekretaris Penguji	:	Dr. Ir. M. Amin Hariyadi, M.T NIP. 19670118 200501 1 001	(_____)
Anggota Penguji	:	Ajib Hanani, M.T NIDT. 19840731 20160801 1 076	(_____)

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tanda tangan dibawah ini:

Nama	: Muhammad Fahmi Kurniawan
NIM	: 14650074
Fakultas/Jurusan	: Sains dan Teknologi / Teknik Informatika
Judul Penelitian	: Optimasi *Server Live Streaming* Menggunakan *Microservices* dan *Load Balancer*

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila dikemudian hari terbukti atau dapat dibuktikan Skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang,
Yang Membuat Pernyataan,

Muhammad Fahmi K.
NIM. 14650074

MOTTO

Dalam setiap melakuakan suatu hal sekecil apapun itu tanamkan dalam hatimu

“Don’t Think You’re The Best but Do The Best”

PERSEMBAHAN

Puji syukur kepada Allah Yang Maha Esa atas limpahan rahmat, taufik, dan hidayah-Nya kepada kita semua, Dzat yang menguasai seluruh alam dan seisinya. Sholawat dan salam kita haturkan kepada Nabi Muhammad SAW, keluarganya, dan para sahabatnya. Semoga kita termasuk umat beliau yang taat sehingga kita termasuk golongan orang yang *husnul khatimah* dan mendapatkan syafaatnya serta ditetapkan nikmat iman dan nikmat Islam didalam hati kita.

Kupersembahkan karya sederhana ini kepada kedua orang tuaku yang selalu kuharapkan ridhonya yaitu bapakku Mochammad Syaichu dan ibuku Najawati sebagai sebagian baktiku kepada beliau. Semoga kebaikan, rahmat, dan perlindungan ALLAH SWT senantiasa tercurahkan kepada beliau.

Kupersembahkan juga karya sederhana ini kepada dewan pengasuh Ma'had Uin Malang terutama KH. Chamzawi yang senantiasa memberikan bimbingan, dan teman-temanku semua. Terima kasih banyak atas do'a, dukungan dan motivasi yang telah diberikan sehingga dapat terselesaikannya penelitian ini.

Muhammad Fahmi Kurniawan

KATA PENGANTAR

Segala puji penulis haturkan kepada Allah Yang Maha Esa yang telah memberikan limpahan rahmat, taufiq, dan hidayah sehingga penulis dapat menyelesaikan skripsi dengan judul "Optimasi *Server Live Streaming* Menggunakan *Microservices* dan *Load Balancer* ini.

Dalam Penulisan skripsi ini penulis banyak memperoleh bantuan dari banyak pihak. Baik berupa motivasi, bimbingan, kritik dan saran yang membangun. Oleh karena itu penulis mengucapkan terima kasih juga kepada:

1. Bapak Dr. Cahyo Crysdian selaku dosen wali dan Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN MAulana Malik Ibrahim Malang.
2. Bapak Dr. Ir. M. Amin Hariyadi, M.T selaku dosen pembimbing I.
3. BApak Ajib Hanani, M.T selaku dosen pembimbing II.
4. Seluruh dosen dan staf Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang yang telah memberikan bekal ilmu yang dapat penulis gunakan untuk menyelesaikan skripsi ini.
5. Bapak dan ibu serta keluarga tercinta yang senantiasa memberikan do'a, motivasi, dan pendidikan yang sangat berharga bagi penulis.
6. Teman-teman semua yang juga telah memeberikan do'a, dukungan, dan informasi sehingga penulis dapat menyelesaikan skripsi ini.

Semoga semua yang telah diberikan mereka kepada penulis menjadi amalan yang diterima oleh ALLAH SWT. Penulis menyadari bahwa dalam skripsi ini

masih terdapat kekurangan. Oleh karena itu, penulis dengan senang hati menerima kritik dan saran yang mebangun dari para pembaca. Akhir kata, semoga apa yang kita kerjakan mendapat ridho dari ALLAH SWT.

Malang, 12 Mei 2020

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

ABSTRAK

Kurniawan, Muhammad Fahmi. 2020. Optimasi *Server Live Streaming* Menggunakan *Microservices* dan *Load Balancer*. Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Dr. Ir. M. Amin Hariyadi, M.T, (II) Ajib Hanani, M.T

Kata Kunci: *Server, Live Streaming, Microservices, Docker, Nginx, Load Balancer*

Optimasi server *live streaming* menggunakan *microservices* dan *load balancer* membantu Sistem Administrator PTIPD UIN Maulana Malik Ibrahim Malang dalam mengoptimalkan *server website live streaming*. Untuk mengoptimalkan *server* yang ada, digunakan metode *microservices* untuk membagi *server* menjadi lebih kecil yang saling terhubung. Pembagian tersebut dilakukan dengan membangun beberapa *container docker*. *Server* dibagi menjadi tiga yaitu *server* A yang merupakan *server* tunggal, *server* B dengan 3 kontainer, dan *server* C dengan 10 kontainer. Kemudian beban kinerja *server* dalam merespon *request* juga dibagi menggunakan *nginx load balancer*. Untuk mengetahui tingkat *response time* pada *server* dilakukan pengujian menggunakan *Apache Benchmark* pada 3 kondisi. Hasil pengujian menunjukkan bahwa *server B* dan C mampu menangani *request* lebih cepat dan stabil disaat *request* berjumlah lebih banyak yaitu pada kondisi 2 dan 3. *Server* B mampu menangani *request* dengan kecepatan rata-rata *response time* 33.95 ms pada kondisi 2 dengan nilai standar deviasi 6.29 ms dan 35.04 ms pada kondisi 3 dengan nilai standar deviasi 5.57 ms. *Server* C dengan kecepatan rata-rata 34.29 ms pada kondisi 2 dengan nilai standar deviasi 6.06 ms dan 33.94 ms pada kondisi 3 dengan nilai standar deviasi 8.85 ms. Sedangkan pada kondisi 1 *server* C jauh lebih cepat dibandingkan dengan server lainnya yaitu dengan kecepatan rata-rata *response time* mencapai 34.68 ms dengan nilai standar deviasi 6.81 ms.

ABSTRACT

Kurniawan, Muhammad Fahmi. 2020. *Live Streaming Server* Optimization Using *Microservices* and Load Balancers. Thesis. Department of Informatics, Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University of Malang. Supervisor: (I) Dr. Ir. M. Amin Hariyadi, M.T, (II) Ajib Hanani, M.T

Keywords: Server, Live Streaming, Microservices, Docker, Nginx, Load Balancer

Optimization of live streaming server using microservices and load balancer helps PTIPD Administrator System of UIN Maulana Malik Ibrahim Malang to optimize live streaming website server. Optimize the existing server, the microservices method used to divide the server into smaller, interconnected servers. The division is building by several container dockers. The server divided into three servers: server A, a single server, server B with three containers, and server C with ten containers. Then, the burden of server performance in responding to requests is also shared using the Nginx load balancer. To determine the level of response time on the server, testing done using Apache Benchmark in 3 conditions. The test results show that servers B and C can handle requests faster and more stable when there are more requests in conditions 2 and 3. Server B can handle requests with an average response time speed of 33.95 ms. In condition 2, a standard deviation value is 6.29 ms and 35.04 ms. Then, in condition 3, a standard deviation value is 5.57 ms. Server C, with average speed, is 34.29 ms. In condition 2, with a standard deviation of 6.06 ms and 33.94 ms, condition 3 with a standard deviation of 8.85 ms. Whereas on condition 1, server C is faster than the other servers with an average response time of 34.68 ms with a standard deviation value of 6.81 ms.

الملخص

كورنياوان، محمد فهمي .2020. التحسين ***Server Live Streaming*** باستحدام ***Microservices*** و ***Load Balancer*** . البحث الجامعي. الشعبة الهندسة المعلوماتية. كلية العلوم والتكولوجيا. جامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج. المشرف: (1) الدكتور محمد آمين هاريبادي (2) عجيب هناني الماجستير.

الكلمات الرئيسية: *Server, Live Streaming, Microservices, Docker, Nginx, Load Balancer*

تحسين *server live streaming* باستحدام *Microservices* و*Load Balancer* تساعد مدير النظام PTIPD بجامعة مولانا مالك إبراهيم الإسلامية الحكومية بمالانج في تحسين *server website live streaming*. لتحسين الملقم فيها، استحدم منهج *microservices* لتقسيم الخادم أصغر ومترابطة. كان التقسيم صنع لبناء *container docker*. إنقسم الملقم إلى ثلاثة أشياء وهم الملقم الألف يعني الملقم الواحد، والملقم الباء بثلاثة إناء، والملقم الجيم بعشرة إناء. ثم تحميل أداء الخادم في إجابة االمطالبة تنقسم باستحدام *nginx load balancer*. لتعريف طباقة *response time* في الملقم فهناك الاحتبار في استحدام *Apache Benchmark* الى ثلاثة أحوال. ونتائج الاحتبار تدل أن الملقم الألف والجيم قادران على التعامل أسرع وثبات حينما أكثر الطلبات في وضع الثاني والثالث. الملقم الباء قادر على التعامل الطلبات بسرعة المتساوي *response time* 33.95 مللي ثانية في وضع الثاني بقيمة الانحراف المعياري 6.29 مللي ثانية و 35.04 مللي ثانية في وضع الثالث بقيمة الانحراف المعياري 5.57 مللي ثانية. الملقم الجيم بسرعة المتساوي 34.29 مللي ثانية في وضع الثاني بقيمة الانحراف المعياري 6.06 مللي ثانية و 33.94 مللي ثانية في وضع الثالث بقيمة الانحرف المعياري 8.85 مللي ثانية. وفي وضع الواحد الملقم الجيم أسرع بنسبة على ملقم الأخر يعني بسرعة المتساوي *response time* تأدي إلى 34.68 مللي ثانية بقيمة الانحراف المعياري 6.81 مللي ثانية.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Media *live streaming* merupakan sarana komunikasi di era digital. *Video* berbasis internet biasa disebut dengan *video on demand (VOD)* sangat mudah diakses oleh pengguna. Di kalangan pelajar, media *live streaming* seakan menjadi kebutuhan pokok. Banyak universitas yang menerapkan perkuliahan secara *online* dengan mengakses sebuah *website live streaming* baik buatan sendiri maupun menggunakan fitur yang telah tersedia dalam media sosial seperti youtube, skype, dll. Dengan adanya media tersebut, kita dapat memanfaatkan teknologi untuk berdakwah dan menyebarkan ilmu pengetahuan sesuai perintah Allah dalam Al-Qur'an Surat Ali Imran Ayat 110 sebagai berikut:

كُنتُمْ خَيْرَ أُمَّةٍ أُخْرِجَتْ لِلنَّاسِ تَأْمُرُونَ بِٱلْمَعْرُوفِ وَتَنْهَوْنَ عَنِ ٱلْمُنكَرِ وَتُؤْمِنُونَ بِٱللَّهِ ۗ

وَلَوْ ءَامَنَ أَهْلُ ٱلْكِتَٰبِ لَكَانَ خَيْرًا لَّهُم ۚ مِّنْهُمُ ٱلْمُؤْمِنُونَ وَأَكْثَرُهُمُ ٱلْفَٰسِقُونَ

Artinya: Kamu adalah umat yang terbaik yang dilahirkan untuk manusia, menyuruh kepada yang ma'ruf, dan mencegah dari yang munkar, dan beriman kepada Allah. Sekiranya Ahli Kitab beriman, tentulah itu lebih baik bagi mereka, di antara mereka ada yang beriman, dan kebanyakan mereka adalah orang-orang yang fasik.

Qur'an Surat Ali Imran Ayat 110 dijelaskan dalam Tafsir Al-Wajiz oleh Syaikh Prof. Dr. Wahbah Az-Zuhaili (Tafsirweb.com, n.d.) bahwa mereka dianggap umat terbaik, karena mereka menyempurnakan diri mereka dengan iman

yang menghendaki untuk melaksanakan segala perintah Allah, dan karena mereka menyempurnakan pula orang lain dengan menyuruh berbuat ma'ruf dan mencegah yang munkar, atau dengan kata lain mengajak manusia kepada Allah, berjihad dan mengerahkan kemampuan untuk mengembalikan mereka dari kesesatan dan kemaksiatan. Dalam ayat ini terdapat seruan halus dari Allah kepada Ahli Kitab untuk mengajak mereka beriman (masuk Islam).

Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang telah membangun sebuah *website live streaming* pada tahun 2017. Tujuan pembuatan *website* tersebut adalah sebagai sarana penyampaian informasi dan ilmu pengetahuan kepada sivitas akademika UIN Maulana Malik Ibrahim Malang serta khalayak umum. *Website* tersebut dibangun dengan menggunakan satu *server*. Jika *website* diakses oleh banyak pengguna, maka beban yang ditanggung oleh *server* sangat berat. Hal tersebut dapat memicu terjadinya *overload* pada jalur koneksi yang menyebabkan *website* ataupun *video* yang diakses mengalami *buffering*. Untuk mengatasi hal tersebut, peneliti akan melakukan optimasi pada *server website live streaming* dengan mengimplementasikan *microservices* dan *load balancer* didalamnya.

Microservice memiliki fitur yang berbeda. Sistem informasi *enterprise* pada umumnya dibangun menggunakan monolitik (tunggal) digeser menjadi terdistribusi. Aplikasi dibagi menjadi beberapa bagian yang memiliki fungsi spesifik dan tidak bergantung pada komponen lainnya (Newman, 2015).

Konsep kunci yang diterapkan dari sudut pandang *microservices* adalah *resilient engineering* (penyekatan). Jika salah satu komponen mengalami kegagalan, maka kinerja komponen yang lain tidak akan terpengaruh. Dengan demikian, komponen yang lain dapat tetap bekerja dengan baik serta masalah yang terjadi dapat terisolasi. Ketika menggunakan sistem tunggal, jika salah satu layanan gagal berjalan, maka semuanya berhenti. Sistem dengan beberapa mesin yang redundan tetap dapat berjalan untuk mengurangi kesempatan kegagalan *(system failure)*. Sebaliknya kegagalan total layanan dapat tertangani dengan menggunakan sistem *microservice* karena layanan fungsionalitas sistem telah tersekat (Namiot, D., 2014).

Docker merupakan metode baru dalam virtualisasi. *Docker* juga tersedia untuk 64-bit Linux. Jika dibandingkan dengan VMware yang merupakan teknik virtualisasi tradisional, sumber daya system pada *docker* lebih ringan serta menawarkan sistem yang dapat digunakan pada laptop Anda maupun pada *cloud* seperti *commit* dan *tag* (Fink, 2014).

Docker dapat digunakan untuk mengembangkan *web server* karena memiliki fitur yang memudahkan proses *deployment* beserta *software* yang dibutuhkan agar tidak menimbulkan konflik antara *web server* dengan yang lainnya (M. Fadlulloh, 2017). Jika menggunakan satu *host web server* dan jumlah *request* dari *client* yang harus ditangani berlebihan maka mengakibatkan terjadinya "*a single point of failure"* (SPOF), yaitu sistem tidak berfungsi karena *server* gagal merespon (Julianto, R., dkk. 2017). Salah satu teknik guna mengatasi masalah tersebut adalah dengan menggunakan teknik *clustering websites*. Teknik ini menggunakan

beberapa virtualisasi beberapa *host* atau *host* dalam satu jaringan. Jika salah satu *host* mati maka *request* yang diterima akan ditangani oleh host yang lainnya sehingga *server* memiliki fungsi *avaibility* yang tinggi (Singh, 2015). Docker memiliki fitur *docker swarm* yang mampu menjalankan *container* pada *multiple hosts*. Hal tersebut memudahkan pengguna untuk mengelola komponen *container* dengan menggunakan *nginx* sebagai *load balancer*.

Dengan mengimplementasikan *microsevices* dan *load balancer* pada *server web live streaming* UIN Maulana Malik Ibrahim Malang, maka *website* tersebut akan lebih ringan jika diakses oleh banyak pengguna. Apabila salah satu *container* bermasalah maka secara otomatis akan dialihkan ke *container* lainnya, sehingga *website* akan tetap dapat diakses oleh pengguna.

1.2 Pertanyaan Masalah

Identifikasi masalah dari penelitian ini adalah seberapa tinggi *response time* pada *server websites live streaming* jika digunakan metode *microservices* dengan *docker* dan *load balancer*?

1.3 Tujuan Penelitian

Tujuan penelitian ini adalah untuk mengukur tingkat *response time* pada *server website live streaming* yang menggunakan metode *microservice* dengan *docker* dan *load balancer*.

1.4 Manfaat Penelitian

Manfaat yang diperoleh dari penelitian ini adalah membantu Sistem Administrator PTIPD UIN Maulana Malik Ibrahim Malang dalam mengoptimalkan *server website live streaming*.

1.5 Batasan Masalah

Batasan masalah yang ada pada penelitian ini adalah sebagai berikut:

1. Penelitian dilakukan di PTIPD UIN Maulana Malik Ibrahim Malang
2. *Webserver live streaming* UIN Maulana Malik Ibrahim Malang

BAB II

STUDI PUSTAKA

2.1 Penelitian Terkait

2.1.1 Penelitian Terkait *Microservices*

Cahyanto (2018) dalam jurnalnya membahas tentang Implementasi Arsitektur *Microsevices* pada *Backend* Comrades yaitu dengan mengubah arsitektur *monolithic* ke *microservices*. Analisis yang dilakukan fokus pada performa yang tercatat dalam *Quality of Services (QoS) Web Services*. Standar dalam QoS adalah *Response Time, Throughput,* dan *Latency*. Pada pengujian performa *web services*, arsitektur *web* yang menerapkan *microservices* pada sistemnya lebih unggul dari *monolithic*.

Hatma (2017) dalam penelitiannya membahas tentang Arsitektur *Microservices* untuk Resiliensi Sistem Informasi. *Clustering docker container* akan diterapkan dengan menggunakan *docker swarm*. Pada sistem *microservices* yang sudah dikembangkan dilakukan tiga macam pengujian yaitu *Unit test, Coponent test, End-to-end test*. Aspek kualitas resiliensi dapat dilihat dari desain arsitektur sistem informasi yang telah dikembangkan. Salah satunya yaitu melakukan pendekatan *microservices* menggunakan *docker container* yang telah dikembangkan dengan sistem *front-end angular2* dan *back-end spring boot*. Cara memudahkan implementasi dan meringankan beban pada *server* dapat dilakukan dengan menerapkan pola komunikasi *REST Service* menggunakan format JSON.

2.1.2 Penelitian Terkait *Docker*

Penelitian tentang Teknik Virtualisasi dalam Pengelolaan Banyak Aplikasi *Web* telah mereview beberapa tulisan mutakhir mengenai teknik virtualisasi, baik berbentuk mesin virtual maupun *container*. Perbandingan keduanya memperlihatkan bahwa teknik *container* merupakan solusi yang tepat dalam pengelolaan banyak aplikasi *web* pada suatu sistem *hosting*. Salah satu *software* yang fokus dalam kontainerisasi yang dilakukan dengan *docker* terbukti mampu menghadirkan aplikasi *website* yang unggul dalam sisi kinerja, *avaibility*, dan dapat menghemat sumber daya *server* (Firmansyah, 2015).

M. Fadlulloh (2017) dalam jurnalnya membahas tentang implementasi *docker* dalam mengelola beberapa *websites*. Kemudahan dalam proses *deployment web* sangat dibutuhkan. Dengan menggunakan *docker*, aplikasi *web* beserta *environment* yang dibutuhkan dapat di *deploy* kedalam satu kontainer. Setiap *container* yang dibuat memiliki IP *private*. *Container* dapat diakses dari luar dengan membuat sebuah *domain* pada tiap *container*. *Nginx* sebagai *reverse proxy* digunakan untuk mengarahkan *domain* ke *container*. Ketika *service web* tiap *container* berjalan dengan baik maka client dapat mengakses semua aplikasi *web*. Menjalankan aplikasi *web* beserta *environment* yang dibutuhkan pada lingkungan *virtual (virtual environment)* dapat meminimalisir timbulnya masalah konflik dependensi.

Jiang & Song (2015) dalam penelitiannya menjelaskan bahwa banyak keuntungan dalam menggunakan *docker*, seperti memudahkan para *developer* dalam konfigurasi *environment*. *Developer* dapat menggunakan *docker* untuk

deployment aplikasi. *Image* dari *docker* dapat dibagikan kepada *developer* lain segingga dapat bekerja pada *environment* yang serupa. Ketika proses pengembangan selesai, *image docker* yang dibuat dapat di-*deploy* pada *server*, sehingga tidak ada permasalahan dalam pengaturan konfigurasi *server*.

2.1.3 Penelitian Terkait *Load Balance*

M. Rexa (2019) dalam penelitiannya membahas *load balance* menggunakan *Docker Swarm*. Terdapat pengelompokan dalam *docker swarm* yang diterapkan yaitu *node manager* dan *worker*. *Node Manager* membawahi beberapa *node worker*. Peneliti mengimplementasikan fungsi deteksi dan *monitoring host down* antar *node*. Jika pengecekan waktu *node worker* ditambah dengan waktu toleransi kurang dari waktu *real node manager* maka *node worker* dianggap *down*. Waktu toleransi merupakan batas waktu yang dibuat oleh peneliti untuk menilai *node worker down*. Jika salah satu *node worker* mengalami *down* maka dapat dilakukan *monitoring* menggunakan *node manager*. Kemudian paket dikirimkan ke *node worker* yang sedang aktif. Penggunaan *load balance* dapat mengurangi *traffic* berdasarkan penggunaan sumber daya *memory*.

E. Yusuf (2013) dalam penelitiannya membahas tentang teknologi *load balance* menggunakan *nginx* dalam mengatasi beban yang diterima *server*. Parameter yang diukur meliputi *response time, request loss, throughput* dan jumlah *request*/s. Pengukuran dilakukan dengan menggunakan aplikasi *httperf* sebagai pembangkit *request* secara simultan untuk mengetahui kemampuan sistem *load balance* dan *server* tunggal dalam melayani *request*. Hasilnya, *load balance* mampu menangani permintaan yang lebih besar daripada *server* tunggal. Server tunggal

mampu menangani *request* maksimal 6,25 *request/s* sedangkan *load balance* mampu melayani *request* maksimal 10,82 per detik sehingga memiliki nilai *throughput* yang lebih bagus dibandingkan *server* tunggal. *Response time network load balancing* lebih cepat serta faktor penyebab *request loss* dapat diminimalisir sehingga mempunyai *availability* yang tinggi dibandingkan *server* tunggal. Kedua *web server* yang diterapkan *load balance* bekerja sama agar *request* yang datang dari *client* dapat dilayani dengan cepat dan ringan.

Rahmad (2017) dalam jurnalnya membahas penggunaan *nginx* sebagai *load balancing* dan *failover*. Pemberian beban diuji menggunakan aplikasi JMeter. Peneliti memonitoring *server* terutama *server backend* dengan bantuan *software monitoring* seperti htop, dan juga beberapa perintah dasar pada linux seperti awk, grep, dan tail. Sistem dirancang dapat meningkatkan *avaibility* dengan melakukan *failover* pada sisi *load balancer* maupun *backend server*. Berdasarkan pengujian menggunakan JMeter, *nginx load balance* tidak dapat meningkatkan nilai *response time* dan *throughput* terhadap *request* laman *html*. Namun *nginx load balancing* mampu meningkatkan *response time* dan *throughput* ketika melayani *request* maupun *input* data menggunakan php ke mysql. Oleh karena itu, penggunaan *load balance* dan *failover nginx* menjadi solusi untuk situs dengan *traffic* tinggi karena mampu menjaga performa dan menstabilkan sistem sehingga tidak terjadi kegagalan baik dalam sistem dan aplikasi *web* yang terpasang.

BAB III

DESAIN DAN IMPLEMENTASI

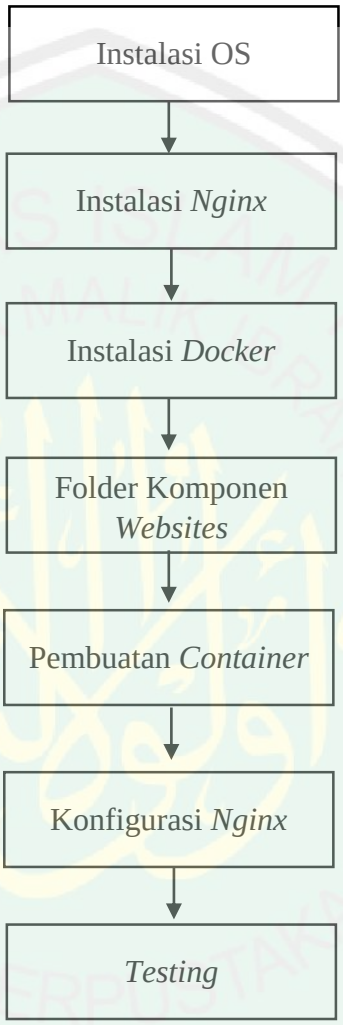
Pada bab ini akan dijelaskan analisa dan perancangan optimasi *server live streaming* menggunakan *microservices* dan *load balancer*.

3.1 Desain Sistem

Desain system pada penelitian ini dapat diuraikan sebagai berikut:

1. Instalasi *operating system* pada VPS yang merupakan *online server*. *Operating system* yang digunakan adalah ubuntu sever 16.04 64bit.
2. Selanjutnya adalah instalasi *nginx* pada *operating* system yang terdapat pada VPS. Nginx berfungsi sebagai *web server yang* merupakan *server* HTTP dan *proxy open source* dengan fungsi sebagai *proxy* IMAP/POP3.
3. Setelah dilakukan instalasi *nginx* maka dilakukan instalasi *docker* pada VPS. *Docker* merupakan *platform open source* yang digunakan untuk membangun, mengirimkan dan menjalankan aplikasi terdistribusi (Docker Docs, 2015). Pada penelitian ini, *docker* digunakan dalam pengimplementasian arsitektur *microservices*.
4. Tahap selanjutnya adalah menyiapkan komponen yang dibutuhkan dalam membangun *websites live streaming* dalam satu *folder*. Komponen tersebut adalah *file* index.html dan *file video* yang akan digunakan untuk *testing* berjalannya *website live streaming*.
5. Setelah menyiapkan komponen kedalam satu *folder* maka dilakukan pembuatan *container docker* dari folder tersebut menggunakan *nginx* sebagai *web server* pada *docker*.

6. Tahap selanjutnya adalah konfigurasi *nginx* yang berada pada *operating system* sebagai *load balancer*. Untuk jelasnya dapat dilihat pada Gambar 3.1 Desain Proses berikut.



Gambar 3.1 Desain Proses

3.1.1 Instalasi *Operating System*

Penelitian ini membangun sebuah *server live streaming* menggunakan *microservices* dan *load balancer*. Sistem operasi yang ditanamkan adalah Ubuntu Server 16.04 64bit karena merupakan sistem operasi *open source* yang memiliki skalabilitas tinggi, lebih stabil, dan terhindar dari virus.

3.1.2 Instalasi Nginx

Web server Nginx merupakan *server* HTTP dan *proxy open source* yang berfungsi sebagai *proxy* IMAP/POP3. *Source code nginx* ditulis oleh salah seorang warga Rusia yang bernama Igor Sysoev pada tahun 2002. *Nginx* dirilis ke publik pada tahun 2004. *Nginx* dapat dikenal karena memiliki beberapa kelebihan diantaranya stabil, minimnya mengkonsumsi sumber daya, dan memiliki tingkat performa yang tinggi (NGINXSoftware.INC, 2014).

3.1.3 Instalasi Docker

Docker merupakan *platform open source* untuk pengelola sistem jaringan dan pengembang perangkat lunak dalam membangun, mengirimkan dan menjalankan aplikasi-aplikasi terdistribusi (Docker Docs, 2015). Pengertian dari definisi tersebut menjelaskan bahwa *docker* merupakan salah satu cara untuk memasukkan sebuah layanan atau beberapa layanan ke dalam lingkungan yang terisolasi bernama kontainer, sehingga layanan tersebut menjadi satu dengan *software* dan pustaka lain yang dibutuhkan (Hane, O., 2015). Pada penelitian ini, *docker* digunakan dalam pengimplementasian arsitektur *microservices*.

3.1.4 Komponen *Websites*

Komponen *website* merupakan file maupun aplikasi yang dibutuhkan dalam membangun sebuah *websites*.Komponen tersebut adalah *file* index.html dan *file video* yang akan digunakan untuk *testing* berjalannya *website live streaming*. Kedua *file* tersebut dikumpulkan kedalam satu *folder* yang akan digunakan untuk pembuatan *container* pada langkah selanjutnya.

3.1.5 Pembuatan *Container*

Pembuatan *container* dilakukan menggunakan *docker* yang merupakan *platform open source* untuk pengelola sistem jaringan dan pengembang perangkat lunak dalam membangun, mengirimkan dan menjalankan aplikasi-aplikasi terdistribusi (Docker Docs, 2015). *Docker* terdiri dari beberapa komponen pembentuk, diantaranya:

1. *Docker Images*

   Merupakan *read-only template* untuk menjalankan sebuah *container*. Sebuah *image* dapat terdiri dari beberapa aplikasi bahkan sebuah sistem operasi. *Images* dapat ditumpuk berlapis-lapis, *image* yang berada paling atas disebut dengan *parent image* dan *image* yang paling bawah disebut dengan *base image*

2. *Docker Container*

   Docker Container merupakan sebuah tempat *(directory)* yang menyimpan segala kebutuhan yang diperlukan agar aplikasi dapat berjalan. Setiap *container* dijalankan dari *docker image* yang telah ditentukan. *Container* dapat dijalankan, diberhentikan, dan dapat dihapus. Setiap *container* yang berjalan terisolasi dalam satu lingkungan dan *platform* aplikasi yang aman, sehingga tidak saling bentrok dengan aplikasi lain walaupun dalam satu *host* yang sama.

3. *Docker Registry*

   Adalah sebuah repositori (*public* atau *private*) yang menyediakan ribuan *docker images*. *Public docker registries* disebut dengan *docker hub. User*

dapat melakukan perintah *push* melalui *docker client* ke *docker registry* untuk penyimpanan dan *sharing*, dan pengguna lain dapat melakukan perintah *pull* untuk mengunduh dan menjalankannya secara langsung.

4. *Docker File*

   Merupakan sebuah *builder* untuk membangun sebuah *image*. *Docker file* adalah dokumen teks atau skrip yang berisi perintah *manual* untuk membangun sebuah *image*. Dengan menggunakan perintah *docker build* dari terminal, *docker* akan membangun *image* secara bertahap berdasarkan]perintah dalam skrip.

5. *Docker Index*

   Docker index terkait dengan *docker hub registry*, meski keduanya memiliki fungsi yang berlainan, docker *index* mengatur *user account, permission, search, tagging* dan hal lain yang tersimpan pada *web interface public*. Ketika melakukan eksekusi perintah *docker*, hal itu digunakan untuk mencari data pada *index* bukan pada *registry*. Ketika menjalankan perintah *docker pull* ataupun *docker push*, *index* akan menentukan apakah diijinkan untuk mengakses atau memodifikasi *image*, dan yang akan menyimpan *image* tersebut setelah mendapatkan hak akses dari *index* adalah *registry*.

3.1.6 Konfigurasi Nginx *Load Balancer*

Nginx, Inc. merilis sebuah *paper* yang menyatakan bahwa *nginx* menyediakan kombinasi yang unik dari *caching proxy, web server,* dan solusi *load balance* untuk sebuah *website*. Fitur *proxy pass* yang berfungsi untuk memindahkan

jalur ke *node* yang ditunjuk terdapat pada *nginx*. *Nginx* juga memiliki *modul load balance*. Metode *load balance* yang terdapat pada *nginx* adalah sebagai berikut:

1. *Round robin*
2. *Least connection*
3. *Weight Round Robin*
4. *IP Hash*

3.1.7 *Testing*

Pengujian *response time* merupakan tahap terakhir dari penelitian ini. *Response time* adalah waktu yang dibutuhkan untuk menyelesaikan satu *request* dan mengirimkannya kembali ke *client*. Pengujian *response time* bertujuan untuk mengukur seberapa cepat suatu *server* dapat menerima *request* dari *client*.

3.2 Implementasi Sistem

Implementasi sistem dalam penelitian ini merupakan tahap transformasi dalam membangun sebuah server yang berdasar pada sub-bab sebelumnya pada *online server* menggunakan VPS dengan spesifikasi sebagai berikut:

1. OS Ubuntu Server 16.04 64bit
2. 1 CPU
3. RAM 1024 MB dan SSD 25 GB
4. *Bandwidth* 1000 GB

3.2.1 Instalasi Operating System

Sistem operasi yang ditanamkan adalah Ubuntu Server 16.04 64bit karena merupakan sistem operasi *open source* yang memiliki skalabilitas tinggi, lebih

stabil, dan terhindar dari virus. Instalasidilakukan secara otomatis oleh penyedia jasa VPS sesuai dengan pilihan sistem operasi yang telah kita pilih untuk dipasang pada VPS.

3.2.2 Instalasi Nginx

Paket Nginx telah tersedia pada repository *default* Ubuntu. Langkah sebelum melakukan instalasi adalah memperbarui indeks paket lokal sistem apt sehingga memiliki akses ke daftar paket terbaru. Kemudian instalasi nginx dapat dilakukan dengan menuliskan *source code* pada terminal ubuntu sebagai berikut:

```
$ sudo apt-get update
$ sudo apt-get install nginx
```

3.2.3 Instalasi Docker

Pada penelitian ini, *docker* digunakan dalam pengimplementasian arsitektur *microservices*. *Docker* akan digunakan untuk membangun beberapa *container* yang berisi *server website live streaming*. Instalasi *docker* dilakukan dengan menuliskan *source code* pada terminal ubuntu sebagai berikut:

```
//1.Update repository
sudo apt-get update
//2.Install CA Certificates
sudo apt-get install apt-transport-https ca-certificates
//3.Tambahkan Key GPG
sudo apt-key adv --keyserver hkp://ha.pool.sks-
keyserver.net:80 --recv-keys \
58118E89F3A912897C070ADBF76221572C52609D
//4.Ganti Repository
echo "deb https://apt.dockerproject.org/repo ubuntu-xenial
main" | sudo tee /etc/apt/sources.list.d/docker.list
//5.Update repository
sudo apt-get update
//6.Memastikan Repository sesuai OS
apt-cache policy docker-engine
//7.Install Linux-image-extra
sudo apt-get install linux-image-extra-$(uname -r) linux-
image-extra-virtual
//8.Install docker
```

```
sudo apt-get install docker-engine
//9.Jalankan Docker
sudo service docker start
```

3.2.4 Komponen *Websites*

Komponen *websites* yang disiapkan yaitu index.html dan videotest.mp4. kedua *file* tersebut dimasukkan dalam satu folder dengan nama web. *Folder* tersebut akan digunakan dalam pembuatan *container*. Isi dari *file* index.html adalah sebagai berikut:

```
<html>
<head>
      <title>Live Streaming A</title>
      <style type="text/css">
      h1{
            font-family: sans-serif;
      }
      </style>
</head>
<body>
      <h1>
            LIVE SERVER A TUNGGAL<br/>

      </h1>

      <video width="720px" height="480px" controls>
            <source src="videotest.mp4" type="video/mp4">
            <source src="mov_bbb.ogg" type="video/ogg">
      </video>

</body>
</html>
```

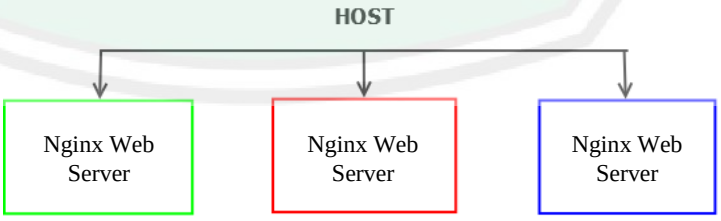
3.2.5 Pembuatan *Container*

Langkah-langkah metode *docker* yang akan dilakukan dalam pembuatan *container* server pada penelitian ini adalah sebagai berikut:

1. Menentukan berapa *container* yang akan dibuat. Pada tanggal 15 Juli 2019, viewer *live* streaming mencapai 6000. Dalam penelitian kali ini akan dibuat tiga buah *server* dengan kriteria *server* A merupakan *server* tunggal (*single*), *server* B merupakan *server* dengan 3

container yang akan menangani 2000 *viewer* ditiap *server* mengacu pada jumlah viewer terbanyak di tahun 2019. Sehingga apabila terjadi lonjakan viewer dengan jumlah yang sama, server mampu menangani hal tersebut. *Server* C merupakan *server* ketiga dengan jumlah 10 *container*. *Server* C diharapkan mampu menangani *viewer* lebih dari 6000 karena UIN Maulana Malik Ibrahim Malang memiliki agenda rutin yang disiarkan melalui *live streaming*. Agenda tersebut biasanya diakses banyak orang baik dari kampus lain maupun alumni seperti kegiatan lomba, khotmil qur'an, dan kegiatan ma'had utamanya.

2. Menyiapkan aplikasi yang akan dijalankan pada setiap kontainer. Penelitian ini akan membangun *server websites*, oleh karena itu, aplikasi yang akan dijalankan di dalamnya adalah aplikasi yang berhubungan dengan *websites* yang berada dalam satu *folder* dengan komponen yang dibutuhkan. Gambar 3.2 adalah *container* yang akan dijalankan dan daftar aplikasi yang ter-*install* pada masing-masing *container* pada server B, untuk server A dan C sama seperti server B hanya berbeda jumlah *container*.



Gambar 3.2 *Container* server B

3. Membangun *docker image* yang dibutuhkan untuk menjalankan *container* dari satu folder yang telah disiapkan pada langkah

sebelumnya dengan mengimplementasikan *source code* sebagai berikut:

```
docker run -ti -d -P -v
/usr/share/nginx/serverA:/usr/share/nginx/html --name
serverA 8960:80 nginx:latest
```

Keterangan :

docker run : untuk menjalankan *container* pada *docker*

-ti : sebuah *option* jika kita eksekusi maka akan langsung masuk kedalam *docker* yang kita buat

-d : menjalankan *container* secara *running background*

-P : publikasikan *container port* ke *host* (8960:80)

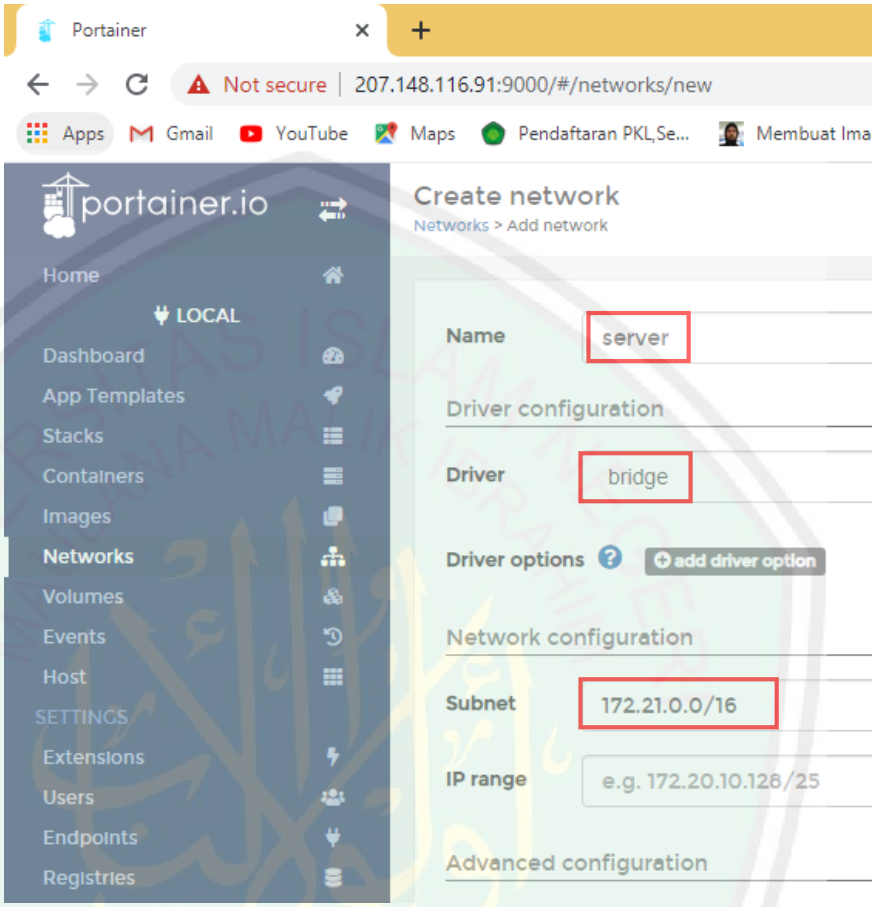
-v : *sharing directory* data

--name : nama server

nginx:lastest : *image* yang akan dijalankan

4. Membuat jaringan dengan *subnet* 172.21.0.0/16 pada *docker* untuk memudahkan memberikan IP pada tiap *container* agar berada dalam satu jaringan yang sama.

Hal tersebut dapat diatur pada *docker portainer* seperti Gambar 3.3.



Gambar 3.3 *Configuration Network*

5. Mengatur *IP address* tiap *container*. Pada penelitian kali ini, tiap *container* akan diberi *IP address* sebagai berikut:

   a. *Container* 1 : 172.21.0.2

   b. *Container* 2 : 172.21.0.3

   c. *Container* 3 : 172.21.0.4

   d. *Container* 4 : 172.21.0.5

   e. *Container* 5 : 172.21.0.6

   f. *Container* 6 : 172.21.0.7

g. *Container* 7 : 172.21.0.8

h. *Container* 8 : 172.21.0.9

i. *Container* 9 : 172.21.0.10

j. *Container* 10 : 172.21.0.11

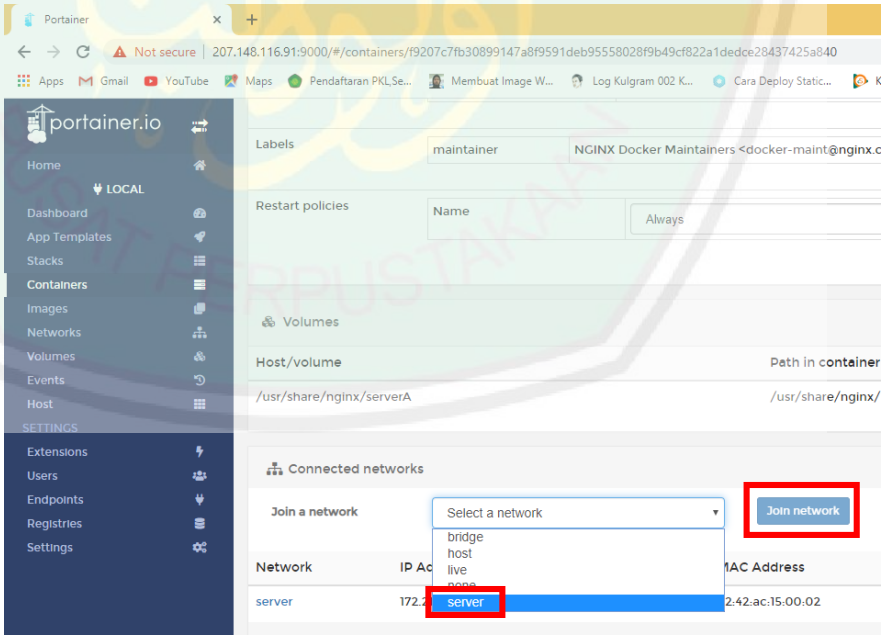
k. *Container* 11 : 172.21.0.12

l. *Container* 12 : 172.21.0.13

m. *Container* 13 : 172.21.0.14

n. *Container* 14 : 172.21.0.15

Pemberian IP dapat dilakukan melalui *docker portainer* pada pengaturan *container*. Kemudian merubah jaringan sesuai dengan nama jaringan yang sudah dibuat sebelumnya sebagaimana Gambar 3.4.



Gambar 3.4 Pemberian *IP Address*

6. Pembagian *container* untuk membangun tiga *server* sebagai berikut:
    a. *Server* A adalah *container* 1.
    b. *Server* B adalah *container* 2 – 4.
    c. *Server* C adalah *container* 5 – 14.
7. Menjalankan *container* yang telah dibuat.
8. Selesai.

3.2.6 Konfigurasi Nginx *load Balancer*

Pada penelitian kali ini, metode *nginx load balance* yang akan diterapkan pada *server* adalah *round robin*. Metode tersebut merupakan salah satu metode penjadwalan paling sederhana. Algortima *round robin* menyalurkan setiap *request* dari *client* yang masuk ke dalam daftar *server* setelahnya secara berurutan hingga kembali ke *server* pertama. Alur penerapan algoritma *round robin* adalah sebagai berikut:

1. Menyiapkan *server* yang akan diterapkan *load balancing* di dalamnya. Pada penelitian kali ini, kita menggunakan tiga *server* (*server* A, B, dan C).
2. Konfigurasi IP pada *source code nginx site available* untuk memanggil *server* sebagaimana telah dibagi pada langkah metode *docker* sebelumnya dengan menambahkan beberapa *source code* di bawah ini pada bagian akhir *file nginx site available*.

```
//source code algoritma round robin load balancing
    //
//Server Tunggal
upstream www.serverA.net{
    server 172.0.21.0.2;
```

```
}
server{
listen 71;
location / {
proxy_pass http://www.serverA.net;
}
}
    //
    //
//Server Dengan 3 Kontainer
upstream www.serverB.net{
    server 172.0.21.0.3;
    server 172.0.21.0.4;
    server 172.0.21.0.5;
}
server{
listen 72;
location / {
proxy_pass http://www.serverB.net;
}
}
    //
    //
//Server dengan 10 Kontainer
upstream www.serverC.net{
    server 172.0.21.0.6;
    server 172.0.21.0.7;
    server 172.0.21.0.8;
    server 172.0.21.0.9;
    server 172.0.21.0.10;
    server 172.0.21.0.11;
    server 172.0.21.0.12;
    server 172.0.21.0.13;
    server 172.0.21.0.14;
    server 172.0.21.0.15;
}
server{
listen 73;
location / {
proxy_pass http://www.serverC.net;
}
}
```

3. Lakukan konfigurasi *link* dari *source code site-available* ke *site-enabled*.
4. *Restart nginx*.
5. Akses dari *browser* ke alamat *IP public server*.

3.2.7 Testing

Pada penelitian kali ini, pengujian *response time* akan dilakukan menggunakan aplikasi *Apache Benchmark* yang merupakan sebuah *tool* untuk menguji performa server. Berikut adalah rumus menghitung *response time* (Oracle, 2010):

$$Response\ time = \frac{n}{r}\ x\ T\ think \quad (1)$$

Keterangan :

n : jumlah pengguna

r : jumlah permintaan per detik yang diterima server

T think : rata-rata waktu berfikir (dalam detik)

BAB IV

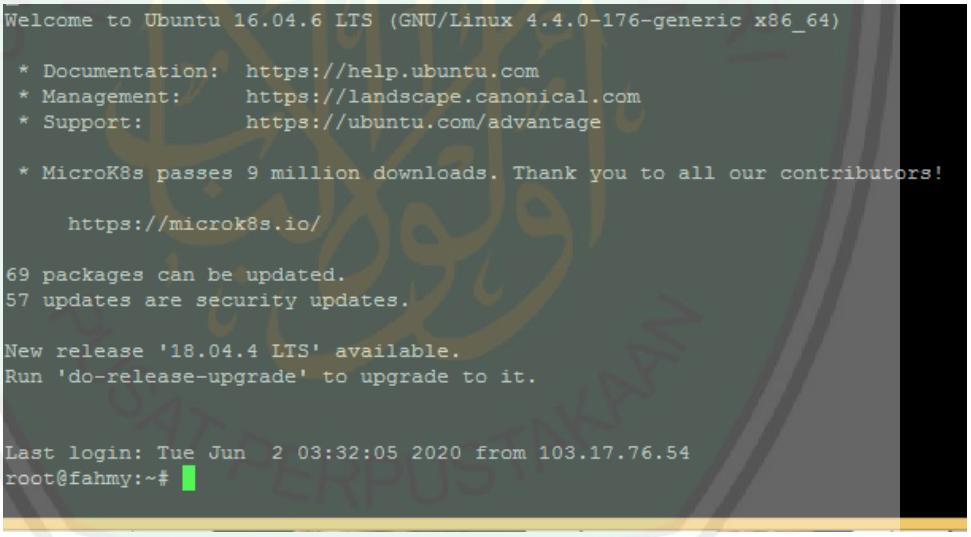
UJI COBA DAN PEMBAHASAN

4.1 Langkah Uji Coba

Langkah-langkah yang dilakukan untuk uji coba server diantaranya sebagai berikut:

1. Instalasi *operating* system

   Instalasi *operating system* dikatakan berhasil dapat dilihat ketika mengakses VPS sebagaimana Gambar 4.1.



Gambar 4.1 Uji *Operating System*

2. Instalasi nginx

   Untuk menguji nginx yang terinstal, maka dapat dilakukan pengecekan versi nginx. Apabila kita dapat mengetahui versi nginx maka instalasi nginx tersebut berhasil. Pengecekan nginx dipat dilakukan sebagaimana Gambar 4.2.

```
root@fahmy:~# nginx -V
nginx version: nginx/1.10.3 (Ubuntu)
built with OpenSSL 1.0.2g  1 Mar 2016
TLS SNI support enabled
configure arguments: --with-cc-opt='-g -O2 -fPIE -fstack-protector-strong
at -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2' --with-ld-opt
Bsymbolic-functions -fPIE -pie -Wl,-z,relro -Wl,-z,now' --prefix=/usr/sha
x --conf-path=/etc/nginx/nginx.conf --http-log-path=/var/log/nginx/access
```

Gambar 4.2 Uji *Nginx*

3. Instalasi docker

   Uji coba instalasi docker dapat dilakukan dengan menjalankan *image* ***hello world*** pada terminal ubuntu sebagaimana Gambar 4.3.

```
root@fahmy:~# sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

root@fahmy:~#
```

Gambar 4.3 Uji *Docker*

4. Komponen websites

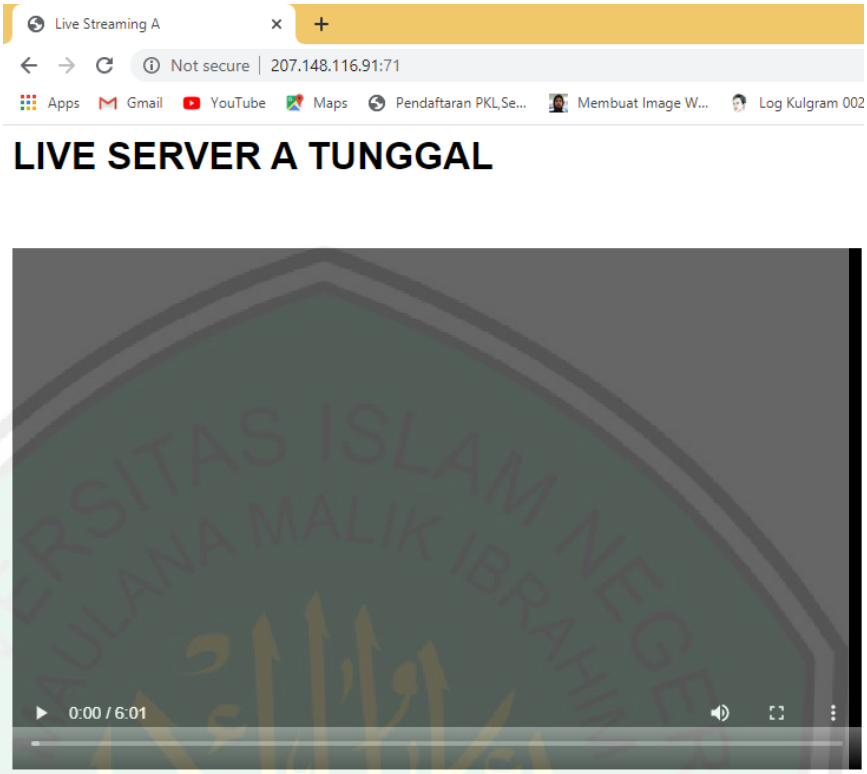
   Pengujian komponen *websites* dilakukan bersamaan dengan uji coba *container* karena *folder* yang berisi komponen *websites* merupakan *folder* untuk membuat *container server*.

5. Pembuatan *container*

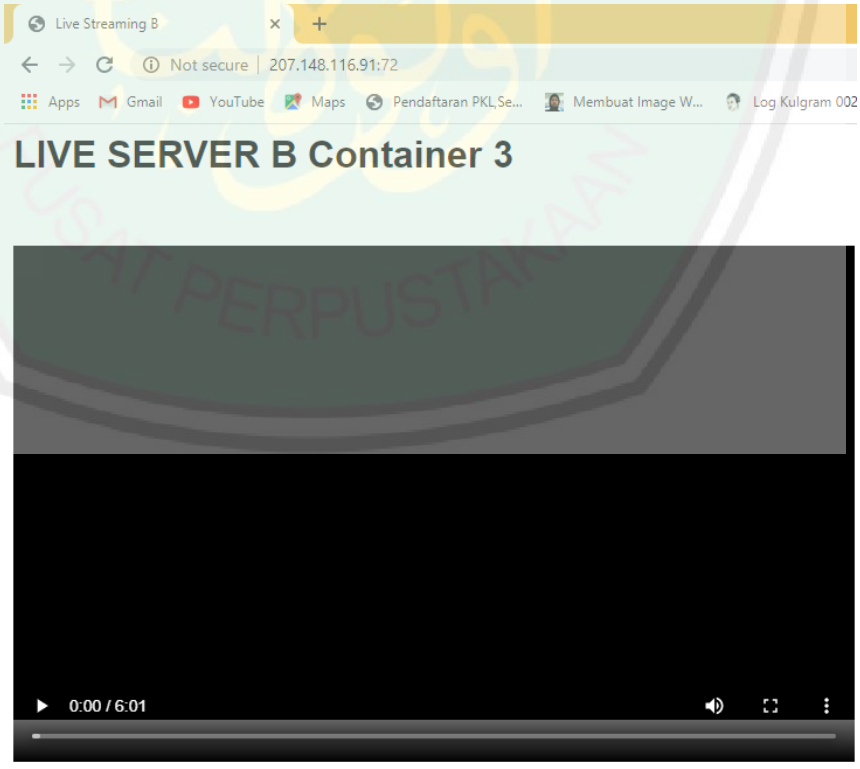
Dalam Penelitian ini, telah dibuat tiga buah server dengan container yang berisikan *nginx web server*. Tiap *container* akan dapat menampilkan *website live streaming*. Container tersebut memiliki *ip addresss* yang berbeda sehingga memudahkan dalam menerapkan *load balance*. *Server* terbagi menjadi 3 bagian dengan *ip address* yang dapat diakses melalui laman internet, yaitu:

a. Server A (207.148.116.91:71) merupakan server tunggal.
b. Server B (207.148.116.91:72) merupakan server dengan 3 buah container.
c. Server C (207.148.116.91:73) merupakan server dengan 10 buah container.

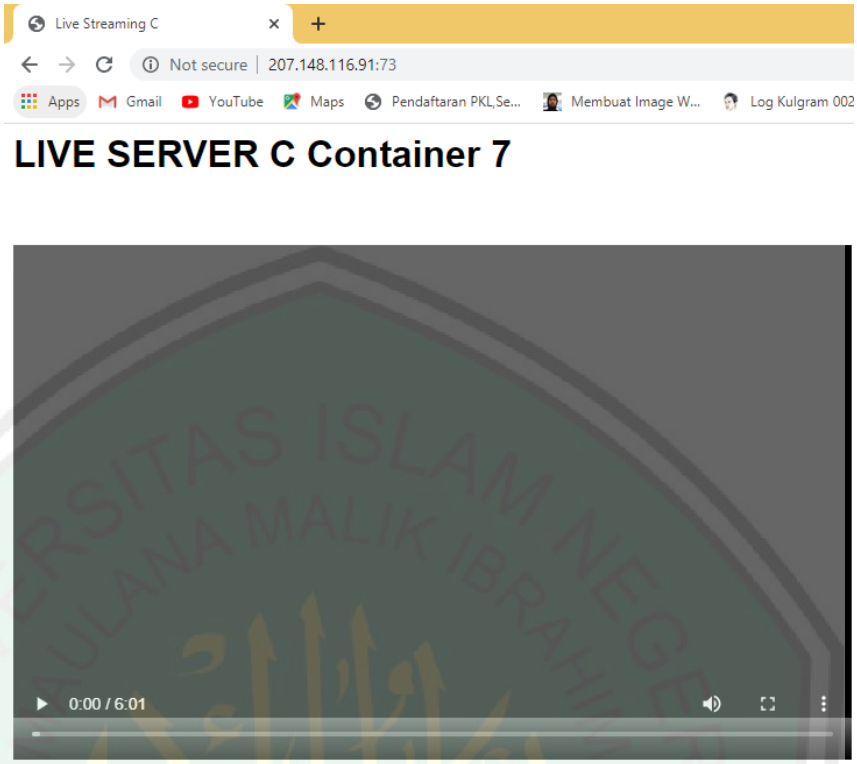
Untuk membedakan *server* yang terakses, maka akan muncul nama *website* dengan nomor *server* beserta nomor *container*. Berikut adalah gambar dari tiap *server*.



Gambar 4.4 *Server* A



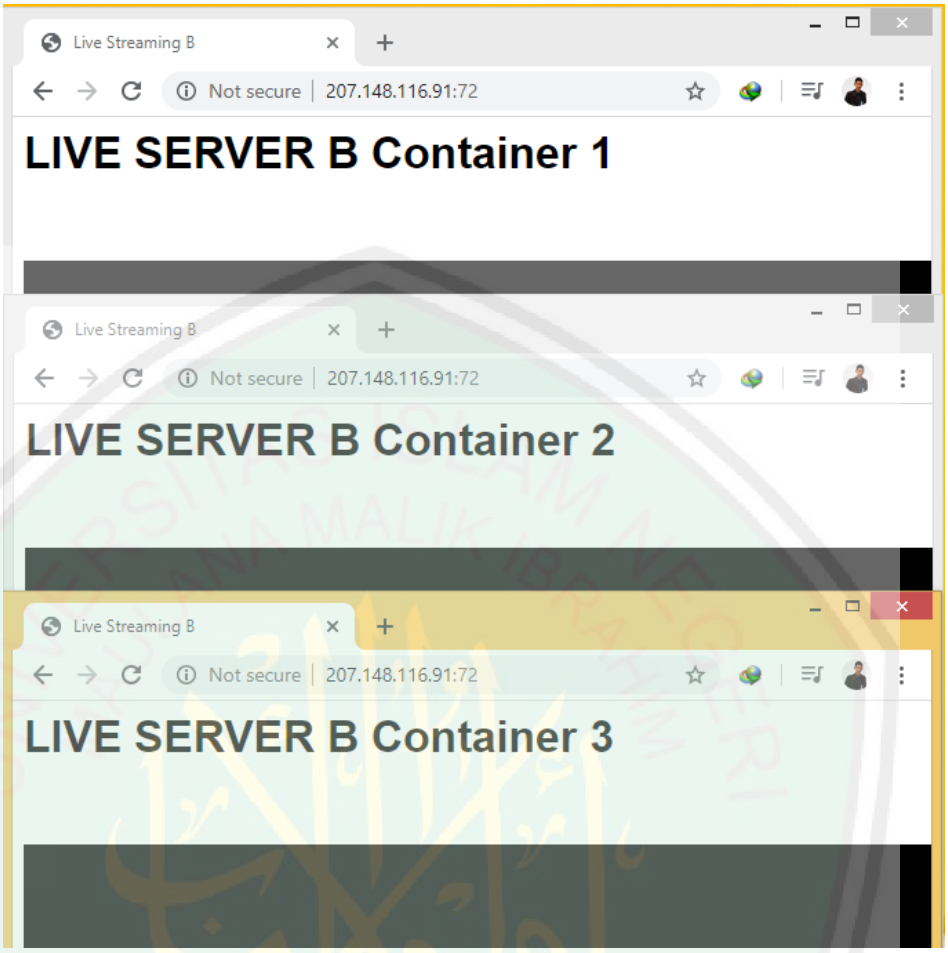
Gambar 4.5 *Server* B



Gambar 4.6 *Server* C

6. Konfigurasi nginx *load balance*

   Metode *load balance* yang diterapkan adalah *round robin*. Metode tersebut akan membagi *request* secara berurutan. Jika kita mengakses *website live streaming* maka akan muncul *website* yang terdapat pada *server* pertama. *Load balance* berhasil jika saat kita *refresh browser*, maka yang muncul adalah *server* kedua. Ketika terdapat *request* baru disaat *server* terakhir berjalan, maka *request* tersebut akan diarahkan ke *server* pertama. Penerapan *load balance* hanya dilakukan pada *server* B dan *server* C. Jadi ketika kita melakukan *refresh browser* atau mengakses *websites* dari perangkat lain maka akan muncul *websites* yang sama namun dari *server* yang berbeda. Hal tersebut dapat dilihat pada gambar pengaksesan *websites* dari 3 *window browser* pada Gambar 4.7.



Gambar 4.7 Uji *Load Balance*

7. Pengujian *response time*

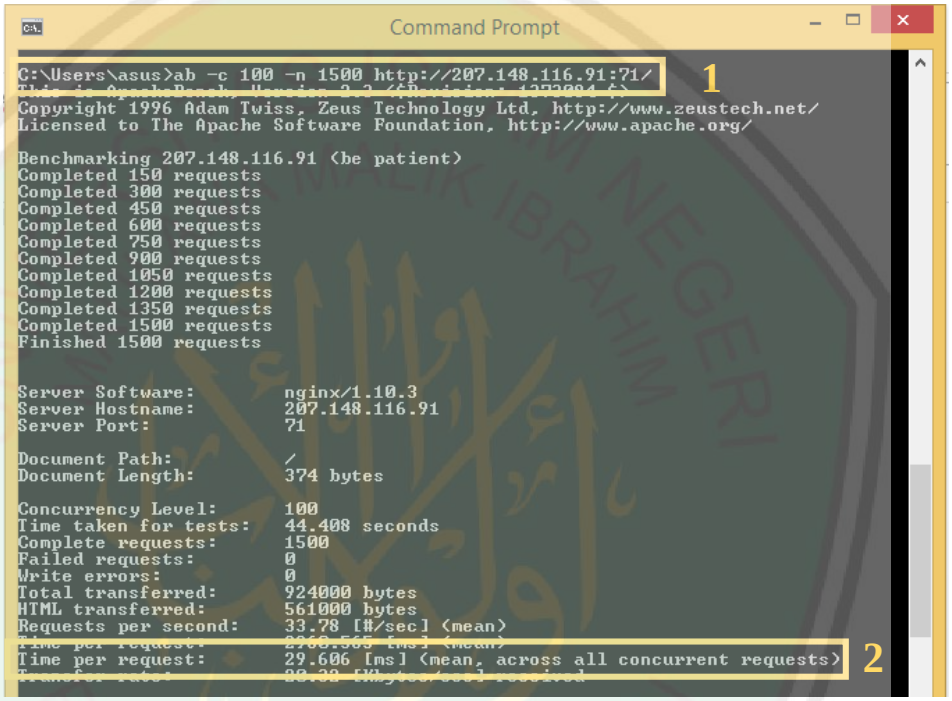
Response time diuji menggunakan *Apache Benchmark* melalui laptop Asus *Core-i3* yang terhubung ke jaringan uin-*wifi*. Hasil pengukuran akan disajikan ke dalam bentuk tabel. Tiap server akan diuji sebanyak sepuluh kali dalam 3 kondisi sebagai berikut:

a. Kondisi 1 (K1) dengan *variable* 100 *c* dan 1500 *n*

b. Kondisi 2 (K2) dengan *variable* 200 *c* dan 2000 *n*

c. Kondisi 3 (K3) dengan *variable* 300 *c* dan 3000 *n*

Variable c adalah jumlah pengunjung yang melakukan *request* dalam waktu bersamaan, sedangkan *variable n* adalah jumlah *request* yang akan dibuat. Di bawah ini adalah salah satu gambar pengujian menggunakan *Apache Benchmark*.



Gambar 4.8 Pengujian menggunakan *Apache Benchmark*

Dari gambar 4.4 kotak dengan angka 1 merupakan *source code* untuk tes performa *webserver*, sedangkan kotak dengan angka 2 merupakan hasil waktu yang dibutuhkan untuk merespon *request*.

Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* A kondisi 1.

Tabel 4.1 Server A Kondisi 1

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	32.84	39.03	-6.19	38.36
2	35.68	39.03	-3.35	11.22
3	35.28	39.03	-3.75	14.03
4	35.15	39.03	-3.88	15.06
5	30.81	39.03	-8.21	67.48
6	33.74	39.03	-5.29	27.96
7	33.30	39.03	-5.72	32.77
8	30.93	39.03	-8.10	65.61
9	34.30	39.03	-4.72	22.32
10	32.72	39.03	-6.31	39.80
11	32.56	39.03	-6.46	41.79
12	56.65	39.03	17.62	310.44
13	45.46	39.03	6.43	41.39
14	43.75	39.03	4.72	22.24
15	44.37	39.03	5.34	28.51
16	40.71	39.03	1.68	2.82
17	46.90	39.03	7.87	61.97
18	43.46	39.03	4.43	19.62
19	43.84	39.03	4.81	23.11
20	48.13	39.03	9.10	82.76

Dari tabel 4.1 maka dapat diketahui:

$$\sum_{i=1}^{20} (x_i - \bar{x})^2 = 969{,}27\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20} (x_i - \bar{x})^2}{n-1} = \frac{969{,}27}{20-1} = 51{,}01\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{969{,}27}{20-1}} = \sqrt{51{,}01} = 7{,}14\ ms$$

Dari tabel dan perhitungan standar deviasi *server* A kondisi 1 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 39.03 ms dengan nilai standar deviasi 7.14 ms. Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* A kondisi 2.

Tabel 4.2 Server A Kondisi 2

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	32.51	41.08	-6.52	42.49
2	32.21	41.08	-6.82	46.56
3	31.67	41.08	-7.35	54.09
4	31.27	41.08	-7.76	60.24
5	30.24	41.08	-8.78	77.17
6	33.74	41.08	-5.28	27.93
7	33.21	41.08	-5.82	33.82
8	30.92	41.08	-8.11	65.70
9	33.67	41.08	-5.36	28.76
10	75.86	41.08	36.83	1356.77
11	62.68	41.08	23.65	559.53
12	52.22	41.08	13.19	173.88
13	49.78	41.08	10.75	115.59
14	47.95	41.08	8.92	79.57
15	49.17	41.08	10.15	102.93
16	49.11	41.08	10.08	101.67
17	50.28	41.08	11.26	126.68
18	31.70	41.08	-7.33	53.75
19	31.78	41.08	-7.25	52.62
20	31.56	41.08	-7.47	55.84

Dari tabel 4.2 maka dapat diketahui:

$$\sum_{i=1}^{20}(x_i - \bar{x})^2 = 3215.60\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1} = \frac{3215.60}{20-1} = 169.24\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{969,27}{20-1}} = \sqrt{169.24} = 13.01\ ms$$

Dari tabel dan perhitungan standar deviasi *server* A kondisi 2 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 41.08 ms dengan nilai standar deviasi 13.01 ms.

Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* A kondisi 3.

Tabel 4.3 Server A Kondisi 3

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	50.03	38.51	11.01	121.12
2	55.68	38.51	16.65	277.10
3	54.27	38.51	15.24	232.24
4	40.26	38.51	1.23	1.51
5	47.30	38.51	8.28	68.48
6	41.55	38.51	2.53	6.38
7	40.35	38.51	1.32	1.75
8	38.24	38.51	-0.79	0.63
9	40.50	38.51	1.47	2.16
10	39.34	38.51	0.31	0.09
11	32.28	38.51	-6.75	45.57
12	33.54	38.51	-5.49	30.09
13	31.54	38.51	-7.49	56.04
14	32.46	38.51	-6.57	43.21
15	32.47	38.51	-6.56	43.08
16	31.54	38.51	-7.49	56.11
17	33.65	38.51	-5.38	28.90
18	31.53	38.51	-7.50	56.26
19	31.73	38.51	-7.30	53.26
20	31.89	38.51	-7.14	51.00

Dari tabel 4.3 maka dapat diketahui:

$$\sum_{i=1}^{20} (x_i - \bar{x})^2 = 1174.97\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1} = \frac{1174.97}{20-1} = 61.84\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{1174.97}{20-1}} = \sqrt{61.84} = 7.86\, ms$$

Dari tabel dan perhitungan standar deviasi *server* A kondisi 3 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 38.51 ms dengan nilai standar deviasi 7.86 ms. Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* B kondisi 1.

Tabel 4.4 Server B Kondisi 1

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	47.33	34.35	8.30	68.83
2	33.64	34.35	-5.39	29.02
3	49.99	34.35	10.96	120.19
4	39.98	34.35	0.95	0.91
5	35.28	34.35	-3.75	14.03
6	31.58	34.35	-7.45	55.56
7	33.70	34.35	-5.33	28.40
8	33.00	34.35	-6.03	36.32
9	33.03	34.35	-6.00	35.95
10	34.38	34.35	-4.65	21.64
11	31.34	34.35	-7.69	59.18
12	31.60	34.35	-7.43	55.26
13	32.10	34.35	-6.93	48.08
14	31.31	34.35	-7.71	59.52
15	31.55	34.35	-7.47	55.87
16	31.25	34.35	-7.78	60.48
17	32.08	34.35	-6.95	48.35
18	31.04	34.35	-7.99	63.77
19	31.43	34.35	-7.60	57.76
20	31.35	34.35	-7.68	59.04

Dari tabel 4.4 maka dapat diketahui:

$$\sum_{i=1}^{20}(x_i - \bar{x})^2 = 978.14\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1} = \frac{978.14}{20-1} = 51.48\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{978.14}{20-1}} = \sqrt{51.48} = 7.18\ ms$$

Dari tabel dan perhitungan standar deviasi *server* B kondisi 1 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 34.35 ms dengan nilai standar deviasi 7.18 ms.

Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* B kondisi 2.

Tabel 4.5 Server B Kondisi 2

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	34.57	33.95	-4.46	19.86
2	32.81	33.95	-6.21	38.62
3	41.22	33.95	2.19	4.81
4	38.36	33.95	-0.67	0.45
5	33.34	33.95	-5.69	32.38
6	38.51	33.95	-0.52	0.27
7	37.85	33.95	-1.18	1.40
8	41.82	33.95	2.79	7.80
9	32.72	33.95	-6.31	39.80
10	31.42	33.95	-7.60	57.83
11	31.18	33.95	-7.85	61.59
12	32.45	33.95	-6.58	43.32
13	31.38	33.95	-7.64	58.44
14	31.74	33.95	-7.29	53.18
15	31.39	33.95	-7.64	58.32
16	31.88	33.95	-7.14	51.05
17	31.28	33.95	-7.75	60.12
18	31.72	33.95	-7.31	53.42
19	31.30	33.95	-7.73	59.75
20	31.97	33.95	-7.06	49.83

Dari tabel 4.5 maka dapat diketahui:

$$\sum_{i=1}^{20}(x_i - \bar{x})^2 = 752.23\, ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1} = \frac{752.23}{20-1} = 39.59\, ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{752.23}{20-1}} = \sqrt{39.59} = 6.29\ ms$$

Dari tabel dan perhitungan standar deviasi *server* B kondisi 2 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 33.95 ms dengan nilai standar deviasi 6.29 ms. Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* B kondisi 3.

Tabel 4.6 Server B Kondisi 3

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	44.23	35.04	5.20	27.06
2	38.02	35.04	-1.01	1.01
3	38.40	35.04	-0.63	0.39
4	37.13	35.04	-1.90	3.60
5	38.55	35.04	-0.47	0.23
6	37.58	35.04	-1.45	2.10
7	34.88	35.04	-4.15	17.20
8	34.55	35.04	-4.47	20.02
9	36.39	35.04	-2.64	6.98
10	41.67	35.04	2.65	7.00
11	32.50	35.04	-6.53	42.60
12	31.49	35.04	-7.54	56.89
13	31.43	35.04	-7.60	57.76
14	32.65	35.04	-6.38	40.65
15	31.87	35.04	-7.16	51.30
16	31.92	35.04	-7.11	50.56
17	32.33	35.04	-6.70	44.87
18	31.88	35.04	-7.15	51.08
19	32.00	35.04	-7.03	49.37
20	31.38	35.04	-7.65	58.47

Dari tabel 4.6 maka dapat diketahui:

$$\sum_{i=1}^{20}(x_i - \bar{x})^2 = 589.14\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1} = \frac{589.14}{20-1} = 31.01\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{589.14}{20-1}} = \sqrt{31.01} = 5.57\ ms$$

Dari tabel dan perhitungan standar deviasi *server* B kondisi 3 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 35.04 ms dengan nilai standar deviasi 5.57 ms.

Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* C kondisi 1.

Tabel 4.7 Server C Kondisi 1

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	47.33	34.68	8.30	68.83
2	33.64	34.68	-5.39	29.02
3	49.99	34.68	10.96	120.19
4	39.98	34.68	0.95	0.91
5	35.28	34.68	-3.75	14.03
6	31.58	34.68	-7.45	55.56
7	33.70	34.68	-5.33	28.40
8	33.00	34.68	-6.03	36.32
9	33.03	34.68	-6.00	35.95
10	34.38	34.68	-4.65	21.64
11	32.14	34.68	-6.89	47.50
12	31.53	34.68	-7.50	56.19
13	31.85	34.68	-7.18	51.61
14	31.96	34.68	-7.07	49.97
15	32.12	34.68	-6.91	47.79
16	31.91	34.68	-7.12	50.70
17	32.75	34.68	-6.28	39.40
18	32.22	34.68	-6.81	46.36
19	32.91	34.68	-6.12	37.46
20	32.37	34.68	-6.66	44.39

Dari tabel 4.7 maka dapat diketahui:

$$\sum_{i=1}^{20}(x_i - \bar{x})^2 = 882.19\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1} = \frac{882.19}{20-1} = 46.43\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{882.19}{20-1}} = \sqrt{46.43} = 6.81\ ms$$

Dari tabel dan perhitungan standar deviasi *server* C kondisi 1 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 34.68 ms dengan nilai standar deviasi 6.81 ms. Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* C kondisi 2.

Tabel 4.8 Server C Kondisi 2

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	32.28	34.29	-6.75	45.50
2	43.32	34.29	4.29	18.44
3	42.30	34.29	3.27	10.69
4	36.56	34.29	-2.47	6.11
5	33.92	34.29	-5.10	26.06
6	32.72	34.29	-6.31	39.80
7	34.63	34.29	-4.40	19.37
8	39.08	34.29	0.05	0.00
9	30.99	34.29	-8.03	64.56
10	36.50	34.29	-2.53	6.38
11	31.59	34.29	-7.44	55.36
12	31.68	34.29	-7.35	53.99
13	35.59	34.29	-3.44	11.84
14	32.40	34.29	-6.63	43.94
15	31.77	34.29	-7.26	52.73
16	32.70	34.29	-6.33	40.09
17	31.83	34.29	-7.20	51.82
18	32.53	34.29	-6.50	42.19
19	31.63	34.29	-7.40	54.79
20	31.74	34.29	-7.28	53.07

Dari tabel 4.8 maka dapat diketahui:

$$\sum_{i=1}^{20}(x_i - \bar{x})^2 = 696.75\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1} = \frac{696.75}{20-1} = 36.67\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{696.75}{20-1}} = \sqrt{36.67} = 6.06\ ms$$

Dari tabel dan perhitungan standar deviasi *server* C kondisi 2 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 34.29 ms dengan nilai standar deviasi 6.06 ms.

Berikut adalah tabel hasil pengukuran standar deviasi *response time Server* C kondisi 3.

Tabel 4.9 Server C Kondisi 3

No Uji	Nilai	Rata-rata(ms)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
1	32.85	33.94	-6.18	38.24
2	32.49	33.94	-6.54	42.74
3	32.48	33.94	-6.55	42.94
4	32.28	33.94	-6.75	45.57
5	34.60	33.94	-4.43	19.65
6	33.79	33.94	-5.24	27.46
7	63.88	33.94	24.85	617.49
8	34.48	33.94	-4.55	20.68
9	34.70	33.94	-4.33	18.74
10	31.75	33.94	-7.28	53.04
11	31.43	33.94	-7.60	57.76
12	32.03	33.94	-7.00	48.94
13	31.48	33.94	-7.55	56.97
14	31.38	33.94	-7.65	58.55
15	31.29	33.94	-7.74	59.92
16	31.63	33.94	-7.40	54.71
17	31.43	33.94	-7.60	57.76
18	31.59	33.94	-7.44	55.33
19	31.51	33.94	-7.52	56.58
20	31.71	33.94	-7.32	53.64

Dari tabel 4.9 maka dapat diketahui:

$$\sum_{i=1}^{20} (x_i - \bar{x})^2 = 1486.69\ ms$$

Oleh karena itu dapat dihitung varian sebagai berikut.

$$s^2 = \frac{\sum_{i=1}^{20} (x_i - \bar{x})^2}{n-1} = \frac{1486.69}{20-1} = 78.25\ ms$$

Dari hasil perhitungan nilai varian didapatkan nilai standar deviasi sebagai berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{20}(x_i - \bar{x})^2}{n-1}} = \sqrt{\frac{1486.69}{20-1}} = \sqrt{78.25} = 8.85\ ms$$

Dari tabel dan perhitungan standar deviasi *server* C kondisi 3 dapat kita ketahui bahwa *server* memiliki rata-rata *response* time 33.94 ms dengan nilai standar deviasi 8.85 ms.

Berikut adalah tabel hasil pengukuran *response time*.

Tabel 4.10 *Response Time Server* A

No	Ket	c	n	Response Time (ms)																		Rata-rata ms)	Standar Deviasi
				Uji 1	Uji 2	Uji 3	Uji 4	Uji 5	Uji 6	Uji 7	Uji 8	Uji 9	Uji 10	Uji 11	Uji 12	Uji 13	Uji 14	Uji 15	Uji 16	Uji 17	Uji 18	Uji 19	Uji 20		
1	K1	100	1500	32.84	35.68	35.28	35.15	30.81	33.74	33.3	30.93	34.3	32.72	32.56	56.65	45.46	43.75	44.37	40.71	46.9	43.46	43.84	48.13	33.94	7.14
2	K2	200	2000	32.51	32.21	31.67	31.27	30.24	33.74	33.21	30.92	33.67	75.86	62.68	52.22	49.78	47.95	49.17	49.11	50.28	31.7	31.78	31.56	41.08	13.01
3	K3	300	3000	50.03	55.68	54.27	40.26	47.3	41.55	40.35	38.24	40.5	39.34	32.28	33.54	31.54	32.46	32.47	31.54	33.65	31.53	31.73	31.89	38.51	7.86

Rata-rata *response time server* pada K1 adalah 33.94 ms. *Server* mampu menangani *request* dengan cepat pada uji 5 K1 serta perbedaan hasil uji tidak jauh berbeda. Jika dibandingan dengan K2, pada uji 10 server mengalami penurunan kecepatan *response time* hingga 75.863 ms. Namun apabila melihat hasil uji sebelumnya, kondisi koneksi internet dapat mempengaruhi hasil uji 10. Berbeda dengan hasil uji K3, nilai rata-rata mencapai 38.51 ms. Hal tersebut menunjukkan bahwa *server* mengalami penurunan *response time* jika dibandingkan dengan K1. Jika dilihat dari nilai standar deviasi maka *response time* tercepat adalah saat *server* berada pada K1.

Tabel 4.11 *Response Time Server* B

No	Ket	c	n	Response Time (ms)																		Rata-rata ms)	Standar Deviasi
				Uji 1	Uji 2	Uji 3	Uji 4	Uji 5	Uji 6	Uji 7	Uji 8	Uji 9	Uji 10	Uji 11	Uji 12	Uji 13	Uji 14	Uji 15	Uji 16	Uji 17	Uji 18	Uji 19	Uji 20		
1	K1	100	1500	47.33	33.64	49.99	39.98	35.28	31.58	33.7	33	33.03	34.38	31.34	31.6	32.1	31.31	31.55	31.25	32.08	31.04	31.43	31.35	34.35	7.18
2	K2	200	2000	34.57	32.81	41.22	38.36	33.34	38.51	37.85	41.82	32.72	31.42	31.18	32.45	31.38	31.74	31.39	31.88	31.28	31.72	31.3	31.97	33.95	6.29
3	K3	300	3000	44.23	38.02	38.4	37.13	38.55	37.58	34.88	34.55	36.39	41.67	32.5	31.49	31.43	32.65	31.87	31.92	32.33	31.88	32	31.38	35.04	5.57

Dari tabel 4.10 dapat kita ketahui bahwa pada K1 *server* mengalami penurunan kecepatan *response time* ketika uji 1, 3, dan 4. Rata-rata K1 adalah 34.35 ms. Rata-rata *response time* K2 merupakan yang tercepat dibandingkan dengan lainnya. Namun K2 juga mengalami penurunan kecepatan respon pada uji 3, 4, 6, 7 dan 8. Hasil uji K3 relatif stabil, hal tersebut menunjukkan

bahwa *server* mampu menangani *request* dengan baik. Jika dilihat dari nilai standar deviasi maka *response time* tercepat adalah saat *server* berada pada K3.

Tabel 4.12 *Response Time Server* C

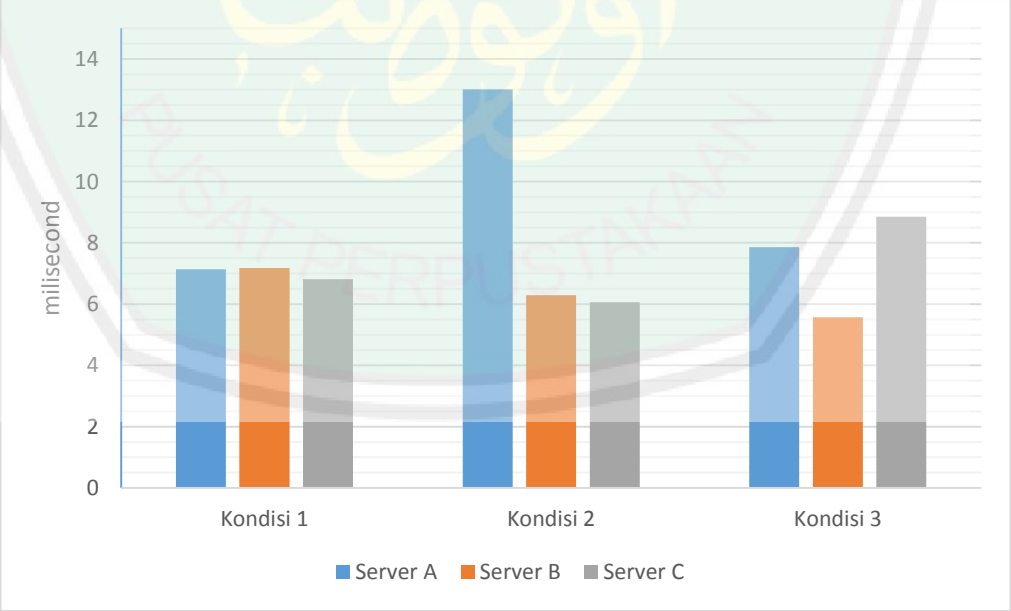
No	Ket	c	n	Response Time (ms)																		Rata-rata ms)	Standar Deviasi
				Uji 1	Uji 2	Uji 3	Uji 4	Uji 5	Uji 6	Uji 7	Uji 8	Uji 9	Uji 10	Uji 11	Uji 12	Uji 13	Uji 14	Uji 15	Uji 16	Uji 17	Uji 18	Uji 19	Uji 20		
1	K1	100	1500	47.33	33.64	49.99	39.98	35.28	31.58	33.7	33	33.03	34.38	32.14	31.53	31.85	31.96	32.12	31.91	32.75	32.22	32.91	32.37	34.68	6.81
2	K2	200	2000	32.28	43.32	42.3	36.56	33.92	32.72	34.63	39.08	30.99	36.5	31.59	31.68	35.59	32.4	31.77	32.7	31.83	32.53	31.63	31.74	34.29	6.06
3	K3	300	3000	32.85	32.49	32.48	32.28	34.6	33.79	63.88	34.48	34.7	31.75	31.43	32.03	31.48	31.38	31.29	31.63	31.43	31.59	31.51	31.71	33.94	8.85

Response time tercepat pada K1 adalah saat uji 12, kondisi 2 terjadi saat uji 9, sedangkan pada K3 yaitu saat uji ke 10. Dari tabel 4.11 dapat kita lihat bahwa hasil rata-rata *response time server* pada ketiga kondisi tidak jauh berbeda walaupun jumlah *request* yang ditangani berbeda. Hal tersebut menunjukkan bahwa dalam ketiga kondisi server mampu menangani *request* dengan baik. Jika dilihat dari nilai standar deviasi maka *response time* tercepat adalah saat *server* berada pada K2.

4.2 Pembahasan

Ada beberapa hal yang dapat menghambat pelayanan *server* terhadap permintaan *client* diantaranya adalah kecepatan *processor*, kecepatan kartu jaringan, dan kapasitas *memory*. Peralatan yang digunakan dalam penelitian ini sangat terbatas, sehingga hal tersebut tidak dapat dihindari.

Secara teori, *processor server* banyak digunakan untuk memproses kode *html* dan lainnya oleh *engine server web* seperti Nginx dan Apache2. Sedangkan *Memory* pada *server web* juga banyak digunakan untuk memproses *query engine server database*. Banyaknya kapasitas *memory* yang digunakan dalam memproses *query* bergantung pada jumlah *query* dan kompleksitas dalam satuan waktu. Berikut ini adalah hasil nilai standar deviasi perhitungan *response time* ketiga *server* dalam bentuk diagram.



Gambar 4.9 Diagram *response time server*

Dari gambar 4.5 dapat kita simpulkan bahwa *server* A pada kondisi 1 lebih cepat dalam merespon *request* dari pada saat kondisi 2. Namun jika dibandingkan dengan kondisi 3, *server* A pada pengujian kondisi 2 mendapatkan hasil lebih lambat dalam merespon *request* yaitu dengan hasil perhitungan standar deviasi kondisi 2 yang memiliki nilai 13.01 ms.

Server B pada pengujian kondisi 3 mampu merespon *request* dengan baik dan lebih cepat dibandingkan dengan kondisi lainnya padahal beban yang diterima *server* pada kondisi 3 lebih berat. Hal tersebut menandakan bahwa *load balancer* yang diterapkan pada *server* bekerja dengan baik karena mampu membagi beban *server* dengan baik sehingga memiliki *respose time* cepat.

Server C pada pengujian kondisi 3 mengalami perlambatan dalam merespon *request*. Hal tersebut dapat dilihat dari nilai standar deviasi *response time server* yang mencapai angka 8.85 ms. Hal tersebut dapat terjadi karena beban yang ditanggung *server* pada kondisi 3 lebih berat dibandingkan kondisi lainnya. Namun tidak menutup kemungkinan perlambatan *response time* server tersebut terjadi karena spesifikasi *server* yang terbatas dan juga koneksi internet dalam melakukan pengujian. Namun secara keseluruhan *server* mampu merespon *request* dengan cepat kurang dari 1 detik. Sebagaimana dapat diamati dari hasil pengujian *server* di atas yang mampu merespon dalam ukuran milidetik.

Dengan *server* yang mampu merespon *request* dengan cepat diharapkan dapat membantu UIN Malang dalam mempublikasikan kegiatan yang ada di kampus secara *real time* terutama dalam hal dakwah keagamaan. Sebagaimana Al-

Qur'an telah menjelaskan bahwa kita dianjurkan untuk saling tolong-menolong dalam penggalan Q.S. Al-Maidah ayat 2 sebagai berikut

...وَتَعَاوَنُواْ عَلَى ٱلۡبِرِّ وَٱلتَّقۡوَىٰۖ ...

Artinya: Dan tolong-menolonglah kamu dalam (mengerjakan) kebajikan dan taqwa.

Penggalan Q.S. Al-Maidah ayat 2 dijelaskan dalam Tafsir Al-Wajiz oleh Syaikh Prof. Dr. Wahbah Az-Zuhaili (Tafsirweb.com, n.d.) bahwa hendaknya sebagian dari kamu membantu sebagian yang lain dalam kebaikan. Kebajikan adalah nama yang mengumpulkan segala perbuatan, baik lahir maupun batin, baik hak Allah maupun hak manusia yang dicintai dan diridhai oleh Allah. Dan takwa disini adalah nama yang mengumpulkan sikap meninggalkan segala perbuatan-perbuatan lahir dan batin yang dibenci oleh Allah dan Rasul-Nya. Seriap perbuatan baik yang diperintahkan untuk dikerjakan atau setiap perbuatan buruk yang diperintahkan untuk dijauhi, maka seorang hamba diperintahkan untuk melaksanakannya sendiri dan dengan bantuan dari orang lain dari kalangan saudara-saudaranya yang beriman, baik dengan ucapan atau perbuatan yang memacu dan mendorong kepada-Nya.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian ini menggunakan tiga buah *server*. *Server* A merupakan *server* tunggal tanpa *load balance*. *Server* B adalah *server* dengan tiga *container* serta *load balance*. Sedangkan *server* C merupakan *server* dengan sepuluh *container* seta *load balance*. Berdasarkan hasil implementasi dan uji coba *server*, maka dapat diambil kesimpulan bahwa *server B* dan C mampu menangani *request* lebih cepat dan stabil disaat *request* berjumlah lebih banyak yaitu pada kondisi 2 dan 3. *Server* B mampu menangani *request* dengan kecepatan rata-rata *response time* 33.95 ms pada kondisi 2 dengan nilai standar deviasi 6.29 ms dan 35.04 ms pada kondisi 3 dengan nilai standar deviasi 5.57 ms. *Server* C dengan kecepatan rata-rata 34.29 ms pada kondisi 2 dengan nilai standar deviasi 6.06 ms dan 33.94 ms pada kondisi 3 dengan nilai standar deviasi 8.85 ms. Sedangkan pada kondisi 1 *server* C jauh lebih cepat dibandingkan dengan server lainnya yaitu dengan kecepatan rata-rata *response time* mencapai 34.68 ms dengan nilai standar deviasi 6.81 ms.

5.2 Saran

Beberapa saran dari penulis sebagai pertimbangan bagi penelitian selanjutnya adalah dalam pengujian *server* memerlukan koneksi internet yang stabil sehingga lebih baik menggunakan kabel LAN untuk menjaga stabilitas internet saat pengujian berlangsung serta menggunakan *server* yang memiliki spesifikasi lebih tinggi.

DAFTAR PUSTAKA

Adiputra, F. (2015). Container dan Docker : Teknik Virtualisasi Ddalam Pengelolaan Banyak Aplikasi Web. *Jurnal SimanteC*, Vol. 4, No. 3 .

Cahyanto Setya Budi, Adam Mukharil Bachtiar. (2018). Implementasi Arsitektur Microservices Pada Backend Comrades. *Repository UNIKOM*.

Docker. (2015). *Linux Container*. Retrieved from http://linuxcontainer.org

Effendi Yusuf, Tengku A Riza, Tody Ariefianto. (2013). Implementasi Teknologi Load Balancer Dengan Web Server Nginx Untuk Mengatasi Beban Server. *Seminar Nasional Teknologi Informasi dan Multimedia STMIK AMIKOM Yogyakarta*.

Fink, J. (2014). Docker : A software as a service, operating system-level virtualization framework. *Code4Lib*, 25.

Hane, O. (2015). *Build Your Own PaaS with Docker.* Packt Publishing.

Harikesh Singh, Dr. Shishir Kumar. (2015). WSQ : Web Server Queueing Algorithm for Dynamic Load Balancing. *Wireless Personal Communication.*

Keyuan Jiang, Quanhao Song. (2015). A Preliminary Investigation of Container-Based Virtualization in Information Technology Education. *Proceedings of the 16th Annual Conference on Information Technology Education* (pp. 149-152). Chicago, Illinois, USA: ACM.

M. Fadlulloh Romadlon Bik, Asmunin. (2017). Implementasi Docker Untuk Pengelolaan Banyak Aplikasi Web. *Jurnal Manajemen Informatika*, 46-50.

Mohamad Rexa Mei Bella, Mahendra Data, Widhi Yahya. (2019). Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host . *Jurnal pengembangan Teknologi Informasi dan Ilmu Komputer*, 3478-3487.

Namiot, D., & Sneps-Sneppe, M. (2014). On micro-services architecture. *International Journal of Open Information Technologies*, 2(9).

Newman, S. (2015). *Building Microservices.* O'Reilly Media, Inc.

Oracle. (2010). Retrieved from https://docs.oracle.com/cd/E19879-01/820-4342/abfch/index.html

Rahmad Dani, Fajar Suryawan. (2017). Perancangan dan Pengujian Load Balancing dan Failover Menggunakan Nginx. *Jurnal Ilmu Komputer dan Informatika*, 43-50.

Riski Julianto, Widhi Yahya, Sabriansyah Rizqika Akbar. (2017). Implementasi Load Balancing Di Web Server Menggunakan Metode Berbasis Sumber Daya CPU Pada Software Defined Networking. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, Vol 1 No 9.

Software.INC, N. (2014, Januari 1). Retrieved from https://www.nginx.com/blog/

Suryotrisongko, H. (2017). Arsitektur Microservice untuk Resiliensi Sistem Informasi. *OAJIS (Open Access Journal of Information System)*, 235-250.

Tafsirweb.com. (n.d.). Retrieved from Tafsirweb: https://tafsirweb.com/1242-quran-surat-ali-imran-ayat-110.html

Tafsirweb.com. (n.d.). Retrieved from Tafsirweb: https://tafsirweb.com/1886-quran-surat-al-maidah-ayat-2.html