Implementasi Distributed Database Pada Learning Management System Menggunakan Platform Redhat Openshift

Bryant Plaudo Santoso, Agustinus Noertjahyana, Justinus Andjarwirawan
Program Studi Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-Mail: bryantplaudo75@gmail.com, agust@petra.ac.id, justin@petra.ac.id

ABSTRAK

Penggunaan teknologi saat ini sudah mulai merambah ke dunia pendidikan, salah satunya adalah penggunaan *E-Learning*. Dalam *learning management system*, tenaga pengajar dapat memberikan *assignment*, materi, tes maupun kuis kepada muridnya. Tetapi jika banyak orang yang mengakses dapat menyebabkan masalah pada *server*. Jika *server down* saat sedang ujian maka akan menjadi masalah yang serius karena para murid tidak akan bisa mengakses *server*. Untuk mengatasi hal ini maka dibutuhkan beberapa *web server* yang siap untuk melayani user agar komputasi tidak hanya berfokus pada sebuah *web server*.

Penelitian ini akan melakukan uji coba dengan implementai *distributed database* pada *learning management system*. Aplikasi akan dijalankan pada *Openshift*. *Distributed database* akan menggunakan MySQL Cluster dengan menggunakan metode *sharding* yang dapat membuat data menjadi beberapa partisi dan disimpan di beberapa *database server*.

Dengan implementasi *distributed database*, diharapkan dapat meningkatkan ketersediaan akan aplikasi. Sehingga ketika terdapat *database server down*, aplikasi tetap dapat dijalankan dengan baik. Selain itu hal ini juga dapat meminimalisir *database server* menjadi *overload* karena diakses oleh banyak *user*.

Kata Kunci: Openshift, Distributed Database. MySQL Cluster

ABSTRACT

The use of technology now has begun to spread to the world of education. One of them is the use of E-Learning. In a learning management system, instructors can give assignments, materials, tests, or quizzes to students. But if many people access it, it can cause problems on the server. If the server goes down during an exam, it will be a serious problem because students will not be able to access the server. To overcome this, we need several web servers that are ready to serve users so that computing is not only focused on a web server.

This research will test the implementation of distributed database on learning management system. The application will run on Openshift. Distributed databases will use MySQL Cluster by using sharding method that can make data into multiple partitions and stored on multiple database servers.

With the implementation of distributed database, it is expected to increase the availability of applications. So when there is a database server down, the application can still be run properly. In addition, this can also minimize the database server to be overloaded because it is accessed by many users.

Keywords: *Openshift, Distributed Database, MySQL Cluster*

1. PENDAHULUAN

Penggunaan teknologi saat ini sudah mulai merambah ke dunia pendidikan, salah satunya adalah penggunaan *E-Learning*. Pemanfaatan *E-Learning* menjadikan proses kegiatan belajar mengajar bisa dilakukan dimana saja, tanpa harus selalu berada di dalam kelas. Dalam pembuatan *E-Learning* dibutuhkan platform *Learning Management System*. Dalam *learning management system* tenaga pengajar dapat memberikan *assignment*, materi, tes maupun kuis kepada muridnya.

Dengan melihat fitur yang ditawarkan, sangat memungkinkan jika *learning management system* digunakan untuk melakukan ujian. Tetapi jika banyak orang yang mengakses dapat menyebabkan masalah pada *server*. Jika *server down* saat sedang ujian maka akan menjadi masalah yang serius karena para murid tidak akan bisa mengakses *server*. Untuk mengatasi hal ini maka dibutuhkan beberapa *web server* yang siap untuk melayani user agar komputasi tidak hanya berfokus pada sebuah *web server*. Untuk membuat beberapa *web server* tentu membutuhkan banyak biaya.

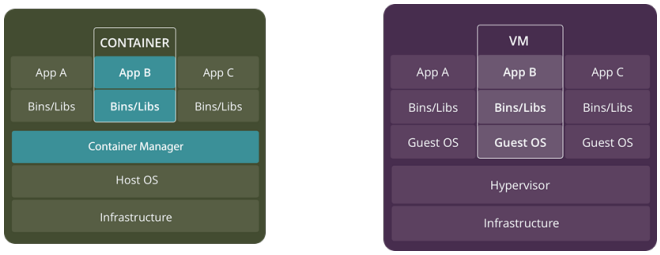
RedHat Openshift adalah sebuah *platform* yang dibuat untuk memudahkan pembuatan, pengembangan, dan menjalankan aplikasi menggunakan proses *containerisation*. Dengan penggunaan platform RedHat Openshift dapat memangkas biaya pemeliharan, biaya *hardware*, dan juga meningkatkan kemampuan untuk skalabilitas. Dengan penerapan distributed database pada sebuah aplikasi yang diletakkan di *Openshift*, diharapkan dapat menghindari *single point of failure*. Sehingga ketika ada satu atau beberapa server yang down diharapkan aplikasi masih terus dapat diakses dan digunakan oleh *user*.

2. DASAR TEORI

2.1 Container

Container memberikan cara standar untuk mengemas aplikasi, konfigurasi, dan dependensi menjadi satu objek. *Container* berbagi sistem operasi dengan server dan dijalankan sebagai proses yang mengisolasi sumber daya, memastikan penyebaran yang cepat, andal, dan konsisten [1].

Gambar merupakan perbandingan infrastruktur dari *container* dengan *virtualisasi*. Container hanya mengisolasi *library*, *dependencies*, dan aplikasi yang dijalankan saja. Hal ini menyebabkan penggunaan *container* bisa lebih ringkas dan lebih ringan. Berbeda dengan virtualisasi yang mengharuskan untuk mengisolasi keseluruhan sistem dimana pada virtualisasi *virtual machine* juga akan membuat kernel tersendiri yang memungkinkan akan diproses langsung pada host OS.



Gambar 1 Perbandingan *container* dengan *virtual machine*

2.2 Docker

Docker adalah sebuah platform *container open source* yang memungkinkan developer membangun, mengemas, dan menjalankan aplikasi dimanapun. Docker memperbolehkan aplikasi untuk berjalan secara independen menggunakan kernel yang sama dengan sistem operasi yang digunakan. Secara independen berarti memperbolehkan container untuk menjalankan beberapa proses dan aplikasi secara terpisah satu sama lain untuk memanfaatkan infrastruktur dengan lebih baik [7].

Docker menyediakan deployment model berbasis image yang dapat memudahkan untuk berbagi aplikasi, atau serangkaian layanan, dengan semua dependensinya di berbagai *environments*. Salah satu keunggulan Docker adalah modularitas aplikasi, pengguna dapat melakukan *update* atau *repair* sebuah aplikasi tanpa harus mempengaruhi lainnya.

2.3 Openshift

OpenShift adalah platform *container* yang dioptimalkan untuk aplikasi web, memungkinkan membangun, menguji, dan menggunakan aplikasi web tanpa menyediakan dan memelihara server khusus untuk setiap aplikasi [6]. Openshift dapat sepenuhnya berjalan diplatform manapun seperti di tempat kita sendiri, di *cloud*, maupun di layanan *hosting*. Openshift juga dioptimalkan untuk meningkatkan produktivitas pengembang dan mempromosikan inovasi.

Tujuan utama dari Openshift adalah untuk memberikan pengalaman terbaik bagi pengembang dan sysadmin mengembangkan, menyebarkan, dan menjalankan aplikasi. Dengan kata lain, OpenShift adalah sebuah lapisan di atas Docker dan Kubernetes yang membuatnya dapat diakses dan mudah bagi pengembang untuk membuat aplikasi dan platform yang merupakan impian bagi operator untuk menyebarkan konten– masih ada untuk beban kerja pengembangan dan produksi [8].

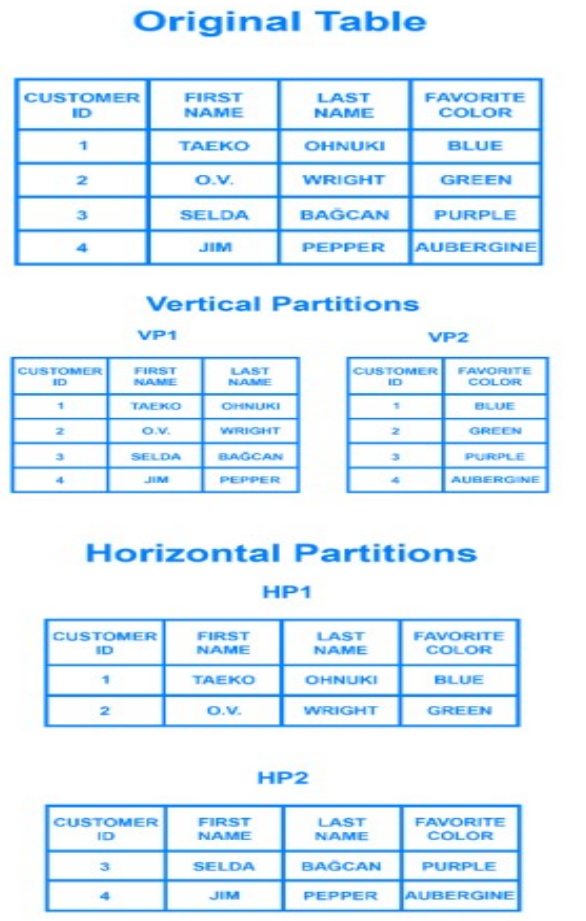
2.4 Load Balancer

Load balancer adalah perangkat yang bertindak sebagai *reverse* proxy dan mendistribusikan *traffic* jaringan dari aplikasi melalui beberapa server. Load balancers digunakan untuk meningkatkan kapasitas (pengguna bersamaan) dan reabilitas aplikasi. Load balancer meningkatkan kinerja aplikasi secara keseluruhan dengan mengurangi beban pada server yang terkait dengan mengelola dan memelihara *session* aplikasi dan jaringan, serta dengan melakukan tugas-tugas khusus aplikasi. Load balancer pada umumnya dikelompokkan ke dalam dua kategori: Layer 4 dan Layer 7. Layer 4 *load balancer* bertindak berdasarkan data yang ditemukan dalam protokol *layer network* dan *transport* (IP, TCP, FTP, UDP). Layer 7 *load balancer* mendistribusikan permintaan berdasarkan data yang ditemukan dalam protokol lapisan aplikasi seperti HTTP [4].

2.5 Database Sharding

Sharding adalah pola arsitektur database yang memisahkan satu tabel menjadi beberapa tabel yang berbeda, atau biasa dikenal juga sebagai partisi. Setiap partisi memiliki skema dan kolom yang sama, tetapi juga baris yang berbeda. Data yang disimpan di masing – masing partisi juga unik dan independen dari data yang disimpan di partisi lain [3]. Sharding terdiri dari 2 yaitu *Horizontal Partitioning* dan *Vertical Partitioning*. Contoh dari pembagian data menggunakan *sharding* dapat dilihat pada Gambar 2.

Horizontal partitioning adalah metode partisi dengan membagi sebuah tabel menjadi beberapa tabel, setiap tabel memliki jumlah kolom yang sama dengan tabel aslinya, tetapi dengan jumlah baris yang lebih sedikit. Sedangkan *Vertical Partitioning* adalah metode partisi yang membagi tabel menjadi beberapa tabel dengan jumlah baris yang sama dengan tabel aslinya, tetapi dengan jumlah kolom yang lebih sedikit.



Gambar 2 Contoh *vertical* dan *horizontal partitioning*

2.6 MySQL Cluster

MySQL Cluster adalah teknologi *database* yang dikembangkan dari mysql konvensional untuk memberikan *high availability storage engine* [5]. MySQL Cluster merupakan tipe database yang dapat beroperasi dalam ukuran data yang besar dan dapat berjalan pada sebuah sistem *share-nothing*. Arsitektur *share-nothing* dapat membuat sistem bekerja dengan *hardware* yang minimal dan tidak membutuhkan perangkat keras ataupun lunak dengan spesifikasi khusus. Arsitektur tersebut juga lebih efektif dan tangguh karena masing – masing komponen memiliki hardisk dan memori tersendiri. MySQL Cluster memiliki tiga tipe node yang masing – masing memiliki peran khusus tersendiri. Tipe – tipe node tersebut adalah:

1. Management Node: Management node adalah node paling ringan pada cluster. Management node memiliki lima peran dalam sebuah cluster dan hanya dibutuhkan pada waktu tertentu. Peran yang pertama yaitu untuk menangani konfigurasi. Kedua untuk mengizinkan node terhubung ke cluster. Ketiga adalah sebagai arbitrase, dimana management node bertugas untuk menentukan data node mana yang harus tetap online jika terjadi split-brain scenario. Keempat adalah untuk fungsi – fungsi administratif yang dilakukan oleh database administrator. Terakhir adalah untuk mempertahankan cluster log dari semua node.

2. Data Node: Data Node adalah inti dari MySQL Cluster, karena node ini merupakan tempat data – data aktual disimpan. Ada dua jenis data node yaitu ndbd dan ndbmt. Ndbd adalah versi single-threaded dari data node, sedangkan ndbmt adalah versi multi-threaded untuk perangkat keras yang lebih modern. Secara fungsional keduanya sama, hanya ada perbedaan dalam hal kinerja.

3. API / SQL Node: API Node adalah tempat dimana query diterima. Setiap API node terhubung ke semua data node dan memiliki akses ke semua data. Meskipun data dilakukan *sharding*, aplikasi ataupun pengguna tidak perlu mengetahui tentang sharding yang terjadi. Ini memungkinkan logika yang lebih sederhana pada aplikasi dan juga melakukan eksekusi query melalu API node yang berbeda. MySQL Cluster mendukung beberapa API seperti NDB API, ClusterJ, Node.Js, NDB-memcached, dan yang paling umum adalah SQL Node (*mysqld instance*).

3. DESAIN SISTEM

3.1 Analisis Masalah

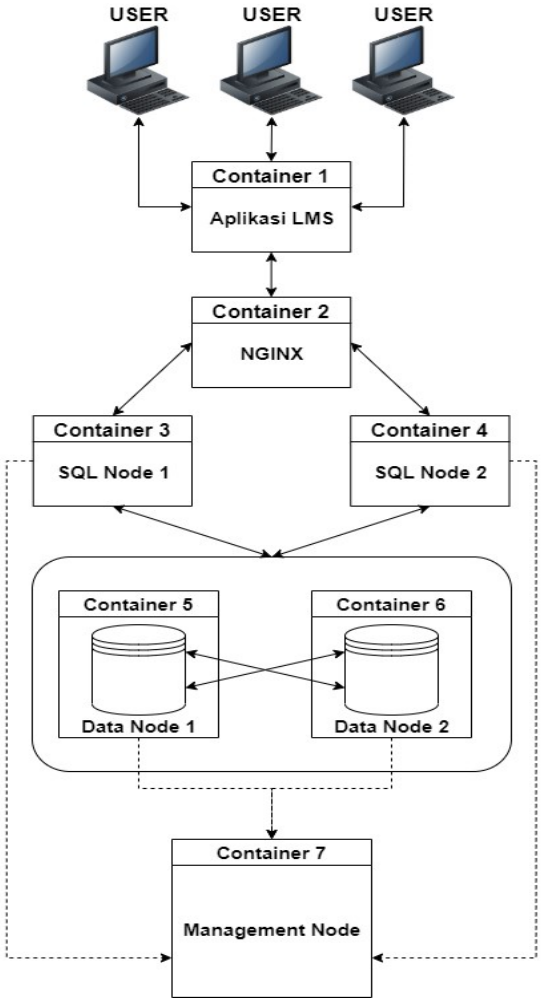
Penelitian mengimplementasikan *distributed database* pada *learning management system* yang dilakukan ini berdasarkan penelitian yang pernah dilakukan sebelumnya oleh mahasiswa Universitas Kristen Petra. Penelitian yang sebelumnya dilakukan yaitu melakukan *scalable* pada *web server*. Jika hanya *web server* yang dilakukan *scalable* dan hanya terhubung dengan sebuah *database server* maka ditakutkan akan terjadi *bottleneck*. Karena dari semua *web server* hanya akan melakukan transaksi ke satu *database server* sehingga tetap memakan waktu yang lama.

Berdasarkan identifikasi masalah tersebut, maka dirasa mengimplementasikan *distributed database* dapat menjadi solusi untuk menghindari *bottleneck*. Dengan adanya *database server* lebih dari satu diharapkan transaksi *read* maupun *write database* dapat dilakukan tanpa harus memakan waktu yang lama. Selain waktu yang lebih singkat, reabilitas dari aplikasi juga dapat ditingkatkan. Karena jika ada *database server* yang mengalami masalah aplikasi masih dapat berjalan.

3.2 Desain Implementasi Sistem

Desain sistem yang dibuat di implementasikan pada sebuah sistem komputer yang terinstal *openshift* sebagai platform *container*. Selanjutnya pada *container* di instal sebuah aplikasi *Learning Management System* yang terhubung ke sebuah sistem *distributed database.* Teknologi *distributed database* yang digunakan adalah MySQL Cluster. Dari aplikasi terhubung kepada sebuah *container* yang terinstal NGINX. NGINX berfungsi untuk membagi *traffic* yang masuk pada *database server*. Algoritma yang digunakan pada NGINX adalah *least connection*. Algoritma *least connection* akan melakukan pembagian traffic ke dua *container* SQL Node yang memiliki jumlah koneksi paling sedikit.

Kemudian dari NGINX dihubungkan ke dua SQL Node yang berguna sebagai konektor untuk aplikasi terhubung ke data node. SQL Node terhubung ke dua buah data node yang berfungsi sebagai tempat penyimpanan data. Setiap sql node dan data node terhubung ke sebuah management node yang berfungsi untuk menghubungkan setiap node menjadi sebuah *cluster*. Pada Gambar 3 merupakan gambaran sederhana dari desain sistem yang akan di implementasikan.



Gambar 3 Desain sistem

4. PENGUJIAN SISTEM

4.1 Proses Pengujian

Proses pengujian dilakukan menggunakan *tools* apache jmeter dan sysbench. Apache jmeter berguna untuk melakukan simulasi user melakukan *request* pada *website*. Dengan jmeter pengguna dapat mengatur berapa total request yang akan dilakukan oleh user. Tools kedua yaitu sysbench berfungsi untuk melakukan *benchmarking* mysql server. Dengan menggunakan sysbench pengguna dapat mendapatkan kemampuan *transaction per second* dari mysql server.

4.2 Hasil Pengujian

Pengujian dilakukan dengan menjalankan empat skenario yang sudah dibuat dengan script recorder seperti pada Tabel 1. Setiap skenario akan di akses oleh satu buah komputer client dengan jumlah request sebesar 50, 100, 200 dan maksimal 300 request. Pengujian akan diulang dengan mencoba menggunakan beberapa node dimatikan. Pengulangan dilakukan sebanyak empat kali dengan rincian pertama 2 sql node dan 2 data node, kedua 1 sql node dan 2 data node, ketiga 2 sql node dan 1 data node, dan terakhir 1 sql node dan 1 data node.

Tabel 1 Skenario Pengujian

No.	Skenario
1	Pengajar Login => Tambahkan Tugas => Logout
2	Pengajar Login => Pilih Tugas => Tambahkan Pertanyaan => Publish Tugas => Logout
3	Pengajar Login => Tambahkan Materi => Logout
4	Siswa Login => Pilih Tugas => Submit Tugas => Logout

Pada Tabel 2, Tabel 3, Tabel 4, dan Tabel 5 dapat dilihat *success rate* dari pengujian setiap skenario dan menggunakan beberapa node server.

Tabel 2 *Sucess rate* Skenario 1

Threads	2 Sql 2 Data	1 Sql 2 Data	2 Sql 1 Data	1 Sql 1 Data
50	100%	100%	100%	100%
100	100%	100%	100%	100%
200	100%	100%	100%	100%
300	100%	100%	100%	100%

Tabel 3 *Success rate* Skenario 2

Threads	2 Sql 2 Data	1 Sql 2 Data	2 Sql 1 Data	1 Sql 1 Data
50	100%	100%	100%	100%
100	100%	100%	100%	100%
200	100%	100%	100%	100%
300	100%	100%	100%	100%

Tabel 4 *Success rate* Skenario 3

Threads	2 Sql 2 Data	1 Sql 2 Data	2 Sql 1 Data	1 Sql 1 Data
50	100%	100%	100%	100%
100	100%	100%	100%	100%
200	100%	100%	100%	100%
300	100%	100%	100%	100%

Tabel 5 *Success rate* Skenario 4

Threads	2 Sql 2 Data	1 Sql 2 Data	2 Sql 1 Data	1 Sql 1 Data
50	100%	100%	100%	100%
100	100%	100%	100%	100%
200	94,57%	89,14%	96,28%	85,71%
300	86,68%	83,33%	84,92%	70,5%

Pengujian berikutnya menggunakan sysbench dilakukan dengan menjalankan *read, write, update*, *and delete* pada 4 tabel yang setiap tabel memiliki 250.000 *record* data selama 300 detik. Hasil pengujian dapat dilihat pada Tabel 6 untuk pengujian dengan 2 sql node dan 2 data node, Tabel 7 untuk pengujian dengan 1 sql node dan 2 data node, Tabel 8 untuk pengujian dengan 2 sql node dan 1 data node, dan Tabel 9 untuk pengujian dengan 1 sql node dan 1 data node.

Tabel 6 TPS 2 Sql 2 Data

No. of Threads	Transaction per Second
50	1129,47
100	1149,76
200	1160,30
300	1141,52
400	1119,43
500	1086,27

Tabel 7 TPS 1 Sql 2 Data

No. of Threads	Transaction per Second
50	845,22
100	833,74
200	819,04
300	824,91
400	798,50
500	793,30

Tabel 8 TPS 2 Sql 1 Data

No. of Threads	Transaction per Second
50	1228,07
100	1257,96
200	1247,83

300	1214,92
400	1179,83
500	1149,75

Tabel 9 TPS 1 Sql 1 Data

No. of Threads	Transaction per Second
50	832,28
100	806,50
200	821,37
300	805,40
400	816,52
500	797,12

5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil pengujian yang dilakukan pada sistem, maka dapat disimpulkan bahwa :

- Dengan mengimplementasikan distributed database MySQL Cluster, aplikasi masih tetap dapat berjalan dengan normal meskipun ada node server yang mati. Meskipun ketika ada server yang mati, tingkat *success rate* menjadi sedikit berkurang.
- Kemampuan CPU lebih berpengaruh daripada penggunaan memory, karena dari pengujian yang dilakukan CPU load meningkat lebih signifikan dibandingkan dengan memory. Pada saat melakukan pengujian *benchmarking* menggunakan sysbench CPU load keseluruhan cluster juga terus naik hingga hampir menyentuh 100%, sedangkan memory load tidak mengalami kenaikan yang signifikan hanya naik sekitar 5% dari saat *idle*.
- Memory usage meningkat saat pertama kali data node dinyalakan, ini disebabkan karena data node melakukan cache seluruh data kedalam memory dan melakukan sinkronasi dengan data node yang lain. Setelah beberapa waktu baru memory berkurang ketika data sudah di tulis kedalam hardisk. Selain itu memory juga meningkat cukup signifikan pada pod yang menjalankan *web server* (aplikasi LMS), karena pada *web server* memory berguna untuk menyimpan temporary data ketika *website* menjalankan beberapa proses dalam waktu yang bersamaan.
- Semakin banyak SQL Node yang digunakan semakin lebih baik, karena dari pengujian yang dilakukan nilai *success rate* dan *transaction per second* ketika ada 2 SQL Node menyala lebih baik ketika hanya 1 SQL Node yang menyala.

5.2 Saran

Saran yang dapat diberikan untuk menyempurnakan dan mengembangkan program ini lebih lanjut antara lain:

- Perhatikan apakah aplikasi bisa diimplementasikan menggunakan MySQL Cluster karena *storage engine* yang digunakan berbeda dengan kebanyakan aplikasi pada umumnya dan memiliki limitasi tertentu.
- Untuk kedepan dapat dilakukan pengujian menggunakan sistem komputer yang memiliki spesifikasi lebih baik dan tidak menggunakan *all-in-one cluster* container. Sehingga diharapkan dapat memberikan hasil *transaction per second* ataupun *success rate* yang lebih baik karena tidak hanya dibebankan pada satu komputer.

6. DAFTAR PUSTAKA

[1] AWS. What is a Container?. URI= https://aws.amazon.com/containers/

[2] Bagui, S., Nguyen, L.T. 2015. Database Sharding: To Provide Fault Tolerance and Scalability of Big Data on the Cloud

[3] Drake, M. 2019. Understanding Database Sharding. URI= https://www.digitalocean.com/community/tutorials/understanding-database-sharding

[4] F5. Load Balancer. URI= https://www.f5.com/services/resources/glossary/load-balancer

[5] Krogh, J.S., Okuno, M. 2017. *Pro MySQL NDB Cluster.* New York: Springer Science+Business Media New York

[6] Lossent, A., Peon, A.R., Wagner, A. 2017. PaaS for web applications with OpenShift Origin

[7] RedHat. What is Docker?. URI= https://www.redhat.com/en/topics/containers/what-is-docker

[8] Shipley, G., Dumpleton, G. 2016. *Openshift for Developers.* California: O'Reilly Media, Inc

[9] Suryanto, S.A., Syaifuddin & Rizqiwati, D. 2018. Pengembangan Mekanisme Akses E-Learning Berbasis Linux Container