Rancang Bangun Layanan Platform as a Service (PAAS) untuk Mendukung Sistem Multi-Tenancy Pengembangan Aplikasi Berbasis Komputasi Awan

Putu Wiramaswara Widya, Royyana Muslim Ijtihadie dan Baskoro Adi Pratomo
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
Email: roy@if.its.ac.id

***Ringkasan*—Tugas Akhir ini merancang sebuah sistem Platform as a Service (PaaS) untuk kebutuhan Web Hosting atau penempatan aplikasi di Internet yang bersifat *multi-tenancy* (multi pengguna) dengan dukungan empat *platform* pengembangan aplikasi Web yaitu PHP 5.5, Node.js, Python 2.7 dan Ruby 1.9; dukungan pembagian muat aplikasi melalui aplikasi penyeimbang muat (*load balancer*) dan mengadopsi tiga prinsip komputasi awan, yaitu: *self-service*, *resource pooling* dan *measured service*. Sistem ini dikembangkan untuk membuka peluang bagi jasa layanan Web Hosting untuk menyediakan layanan Hosting yang lebih kompetitif dengan fitur yang sesuai dengan perkembangan Komputasi Awan saat ini.**

Sistem dikembangkan menggunakan platform pengembangan aplikasi MEAN Framework (MongoDB, Express.js, Angular.js dan Node.js), sistem virtualisasi/isolasi aplikasi berbasis kontainer menggunakan Docker, dan sistem penyeimbang muat berbasis HAProxy.

***Kata Kunci*—Hosting Aplikasi, Platform-as-a-Service, Multi--Tenancy, Komputasi Awan**

I. PENDAHULUAN

Layanan hosting dan pengembangan aplikasi dalam bentuk Web atau sering disebut dengan Web Hosting merupakan layanan yang jamak tersedia di Indonesia.

Web Hosting memiliki berbagai macam jenis [1]. Salah satu jenis yang paling umum adalah Shared Web Hosting menggunakan perangkat lunak bernama cPanel. Pada jenis ini, aplikasi dari banyak pengguna disimpan dalam satu mesin server yang sama. Konsekuensi dari penyimpanan pada server yang sama adalah adanya kemungkinan *bottleneck* baik dari sisi pemrosesan, memori dan jaringan. Selain itu, jenis ini hanya menyediakan satu platform yang sama (Umumnya kombinasi PHP dan MySQL) sehingga tidak cocok digunakan untuk aplikasi Web yang menggunakan platform bahasa pemrograman yang berbeda.

Jenis Hosting selanjutnya adalah Virtual Private Server atau VPS. Layanan ini menawarkan kendali penuh dari sisi pengguna sehingga mereka dapat mengontrol apa saja layanan yang tersedia pada server dari sisi sistem operasi. VPS tersedia dalam berbagai macam jenis layanan berdasarkan jumlah sumber daya penyimpanan memori, jaringan serta kemampuan pemrosesan. Kelemahan dari VPS adalah kerumitannya dalam konfigurasi karena semua konfigurasi dilakukan manual oleh pengguna atau penyewa.

Selain Shared Web Hosting dan VPS, saat ini terdapat jenis web hosting yang mulai populer yaitu Cloud Web Hosting. Layanan ini mengadopsi prinsip komputasi awan yaitu berupa Platform as a Service (PaaS) yang menyediakan jasa platform bagi pengembang Web untuk menempatkan aplikasi mereka di Internet. Layanan ini memberikan fitur tambahan yang menjadi karakteristik komputasi awan: transparansi akses, *multi-platform*, skalabilitas, reliabilitas, keterbukaan akses API dan kemudahan penggunaan. Contoh layanan semacam ini adalah Microsoft Azure, Heroku, OpenShift, Google Apps Engine. Sayangnya, layanan ini masih sangat jarang ada terutama di Indonesia. Kebanyakan layanan yang dipasarkan sebagai cloud web hosting di Indonesia masih menggunakan panel kontrol berbasis cPanel yang memiliki fitur terbatas dan hanya didesain untuk shared web hosting. Pihak ketiga kesulitan menawarkan layanan serupa karena belum ada perangkat lunak PaaS yang didesain khusus untuk kebutuhan hosting banyak penyewa (multi-tenant). Kebanyakan aplikasi PaaS Open Source seperti OpenShift Origin dan CloudFoundry masih difokuskan untuk kebutuhan *private cloud*.

Pada Tugas Akhir ini, dilakukan perancangan perangkat lunak dan sistem jaringan PaaS yang dikhususkan untuk membangun layanan pengembangan dan hosting aplikasi berbasis komputasi awan. Sistem ini mengadopsi beberapa prinsip komputasi awan menurut definisi NIST [3] yaitu: *self-service*, *resource pooling* dan *measured service*. Sistem ini juga mengambil beberapa fitur yang ada pada layanan cloud web hosting seperti OpenShift dan Heroku serta mengadopsi beberapa unsur yang ada pada shared web hosting tradisional seperti sistem penagihan yang sederhana serta kemudahan untuk dibangun oleh pihak ketiga yang ingin membangun layanan sejenis. Sistem dikembangkan menggunakan platform pengembangan aplikasi MEAN Framework (MongoDB, Express.js, Angular.js dan Node.js) dengan sistem isolasi aplikasi berbasis kontainer menggunakan Docker.

II. TINJAUAN PUSTAKA

A. Komputasi Awan

Komputasi awan atau cloud *computing* menurut definisi dari NIST (National Institute of Standard Technology) merupakan model yang memberikan akses segala macam (*ubiquitous*), yang nyaman dan akses jaringan sesuai kebutuhan pada kumpulan sumber daya komputasi yang dapat dikonfigurasi (baik

dari sisi jaringan, *server*, penyimpanan, aplikasi dan layanannya) sehingga bisa ditetapkan secara cepat dengan usaha manajemen seminimal mungkin. [3]

Komputasi awan meliputi komputasi, perangkat lunak, akses data dan layanan penyimpanan yang tidak memerlukan pengetahuan pengguna akhir (end-user) terhadap lokasi fisik dan konfiguasi dari sistem yang diberikan pada layanan. Komputasi awan juga merupakan tren dunia teknologi informasi saat ini yang memindahkan proses komputasi yang awalnya dari komputer desktop atau PC ke perangkat pusat data yang besar. [2]

Tujuan utama dari komputasi awan adalah untuk membuat penggunaan sumber daya terdistribusi menjadi lebih baik dengan mengombinasikannya agar menjadi luaran yang besar serta mampu memecahkan komputasi skala besar. Komputasi awan secara spesifik berhubungan dengan teknologi virtualisasi, skalabilitas, kemampuan inter-operasi, kualitas layanan dan model pelayanan.

Berikut adalah beberapa karakteristik umum yang terdapat pada komputasi awan menurut definisi dari NIST [3]: **On-Demand Self Service** yaitu pelanggan atau penyewa dapat menetapkan kemampuan komputasinya tanpa perlu interaksi secara manual, **Broad Network Access** yaitu kemampuan yang tersedia pada jaringan yang dapat diakses melalui media apapun baik *thin-client* maupun *thick-client*, **Resource Pooling** yaitu kemampuan untuk menyatukan sumber daya komputasi dari penyedia layanan untuk diberikan ke banyak pelanggan atau penyewa dengan sumber daya fisik maupun *virtual* yang secara dinamis diberikan atau dilepas sesuai kebutuhan pelanggan. Dalam hal ini, pelanggan tidak memiliki pengetahuan dan kontrol mengenai dimana sebenarnya suatu sumber daya ditempatkan secara fisik, **Rapid Elasticity** yaitu kemampuan sumber daya yang secara elastis bisa ditetapkan atau dilepaskan, **Measured Service** yaitu memberikan kontrol otomatis untuk mengoptimasi sumber daya dengan meningkatkan pengukuran pada layanan (seperti penyimpanan, pemrosesan dan lebar pita jaringan)

B. Node.js

Node.js merupakan kerangka kerja lisensi terbuka dan lintas platform berbasis Javascript yang digunakan untuk membangun aplikasi sisi *server* dan jaringan. Kerangka kerja ini dibuat pertama kali oleh Ryan Dahl pada tahun 2009 pada situs Web http://nodejs.org. Pada awalnya, Javascript hanya merupakan bahasa pemrograman yang digunakan dan berjalan pada peramban Web. Penggunaan Javascript untuk kebutuhan aplikasi *server* kemudian dikembangkan melalui Node.js sehingga pengembang bisa menulis program aplikasi Web mereka cukup dalam satu bahasa. [4]

Node.js menggunakan mesin Javascript Google V8 seperti yang juga digunakan oleh Google Chrome. Selain itu Node.js juga memberikan kemampuan kepada pengembang untuk membangun aplikasi jaringan yang cepat dan *scalable* menggunakan pola pemrograman berbasis event-*driven* dan *non-blocking I/O* sehingga bisa membuat aplikasi menjadi ringan, cepat dan efisien untuk aplikasi yang bersifat waktu nyata dan intensif data. [5].

C. MEAN Framework

MEAN Framework atau MEAN Stack merupakan kerangka kerja pemrograman yang terdiri dari komponen berikut [7]: MongoDB sebagai perangkat *server* basis data tanpa SQL berbasis dokumen JSON sebagai alternatif dari basis data berbasis SQL yang sudah umum digunakan pada aplikasi bisnis, Express.js merupakan kerangka kerja pengembangan aplikasi Web berbasis Javascript dari sisi *server*, Angular.js merupakan kerangka kerja pengembangan aplikasi Javascript dari sisi klien, Node.js merupakan kerangka kerja pemrograman yang membawahi komponen Express.js.

Kombinasi kerangka kerja pemrograman ini menjadi alternatif dari kombinasi sebelumnya (LAMP, Linux Apache MySQL, PHP) yang sebelumnya sudah mendominasi. Berbeda dengan LAMP yang masih menekankan pada aplikasi Web HTTP request sederhana, MEAN fokus ke penggunaan Javascript berbasis AJAX untuk menambah interaktivitas antara klien dan *server* sehingga pertukaran data dapat dilakukan secara efisien dan tanpa perlu perpindahan anatara halaman. Semua kombinasi perangkat lunak di dalam MEAN menggunakan bahasa pemrograman yang sama, yaitu Javascript, sehingga lebih sederhana dari sisi pemrogram.

D. Docker

Docker merupakan kerangka kerja virtualisasi berbasis sistem operasi Linux 64-bit. Berbeda dengan perangkat lunak virtualisasi lainnya yang melakukan virtualisasi mesin secara penuh, Docker melakukan virtualisasi di atas kernel yang berjalan pada sistem operasi dibawahnya sehingga hanya menggunakan sumber daya memori yang sangat kecil dan dapat berjalan dengan cepat. Docker dikembangkan oleh perusahaan Docker.inc dan dilisensikan dibawah lisensi Apache 2.0.

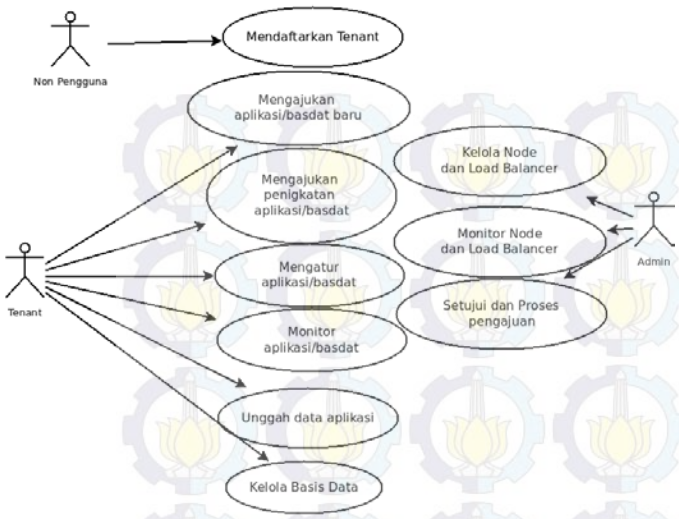
Docker menggunakan istilah *container* untuk setiap mesin *virtual* yang berjalan. Setiap *container* akan diisolasi satu sama lain terhadap sistem operasi utama sehingga tidak akan saling menganggu. Berbeda dengan virtualisasi berbasis *container* lainnya seperti OpenVZ, Docker menawarkan kakas yang mempermudah para pengembang untuk melakukan pengelolaan *container* yang dijalankan di dalam sistem mereka. Docker juga menawarkan situs yang dapat digunakan oleh pengembang untuk mendapatkan citra atau image dari *container* yang sudah siap pakai pada https://registry.hub.docker.com. [8] Selain itu, pengguna juga dapat membangun citra mereka sendiri menggunakan API yang sudah disediakan.

E. HAProxy

HAProxy merupaka perangkat lunak bebas dan terbuka untuk kebutuhan penyeimbangan muat yang membutuhkan ketersediaan *server* yang tinggi. Perangkat lunak ini mendukung pembagian muat untuk akses HTTP maupun yang berbasis socket TCP. Perangkat lunak ini diklaim sebagai standar de-facto untuk aplikasi *load balancer* di sistem operasi Linux maupun pada platform berbasis komputasi awan [9].

III. METODOLOGI

Langkah dan metode yang dilakukan untuk mengerjakan Tugas Akhir ini adalah sebagai berikut.



Gambar 1. Diagram Kasus Penggunaan Sistem

- Penyusunan Proposal Tugas Akhir
- Studi Literatur
- Desain dan Perancangan
- Implementasi Sistem
- Uji Coba dan Evaluasi

IV. Desain dan Perancangan

A. Kasus Penggunaan

Gambar 1 menampilkan kasus penggunaan sistem secara umum.

B. Deskripsi Fitur

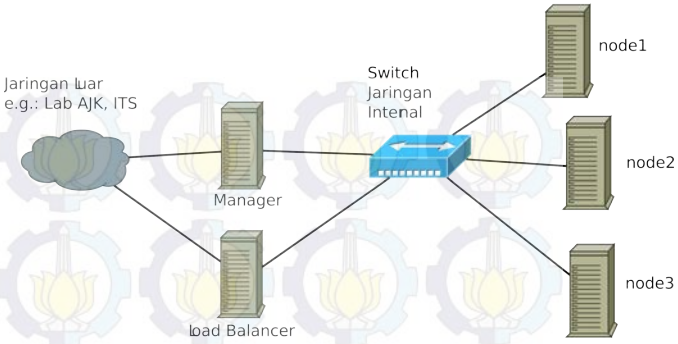
Secara umum, fitur sistem dibagi menjadi dua: fitur untuk penyewa dan fitur untuk administrator.

1) Fitur untuk Penyewa: Berikut adalah fitur yang tersedia untuk penyewa yang sudah terdaftar pada sistem:

- Mengajukan aplikasi Web baru dengan memilih jenis kerangka kerja, kapasitas ruang diska dan jumlah Node yang digunakan.
- Mengajukan basis data baru dengan jenis MySQL berdasarkan kapasitas diska.
- *Scale-Out* untuk menambah Node atau kapasitas diska dari aplikasi Web secara mandiri.
- Mengelola berkas aplikasi Web melalui antarmuka Git.
- Mengelola basis data berbasis *shell* MySQL.
- Menjalankan dan menghentikan aplikasi yang sedang berjalan.
- Memonitor jalannya aplikasi berupa log pesan debug dan log akses.
- Mengubah nama domain aplikasi untuk DNS Server.

2) Fitur untuk Administrator: Berikut adalah fitur yang disediakan untuk Administrator:

- Melakukan verifikasi penagihan yang diajukan oleh pengguna dan aktivasi pesanan dan proses *scaling*.
- Memonitor keadaan di setiap Node.
- Mendata dan mencatat semua Node dan Load Balancer yang terpasang.



Gambar 2. Desain Sistem Secara Umum

C. Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis dan kebutuhan bisnis dan desain dari sistem yang akan dibangun.

1) Desain Umum Sistem: Sistem dibangun dalam beberapa komponen umum yang terkait satu sama lain, yaitu:

- **Manager** sebagai antarmuka pengendali jalannya PaaS dari sisi klien penyewa maupun administrator.
- **Load Balancer** atau penyeimbang muat sebagai titik awal ketika pengguna umum mengakses aplikasi atau halaman Web yang dimiliki oleh penyewa. Komponen ini akan menyeimbangkan beban akses dari suatu aplikasi Web.
- **Node** sebagai *server* yang menyimpan seluruh aplikasi Web dan basis data serta menjalankannya. Aplikasi dapat disimpan secara redundan di beberapa Node sekaligus untuk meningkatkan kemampuan Load Balancer.

Gambar 2 menampilkan diagram arsitektur hubungan antara komponen.

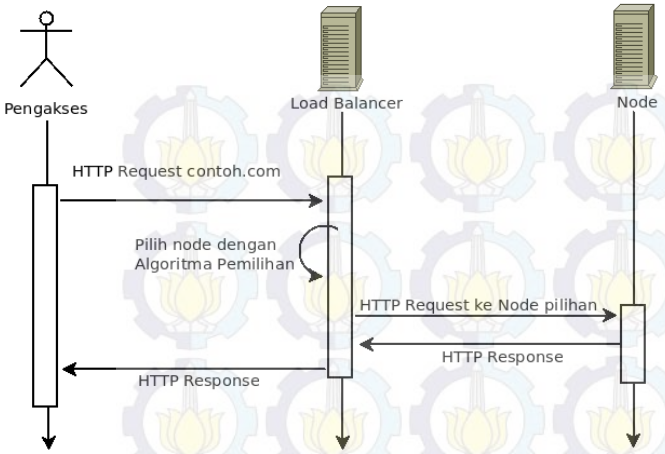
2) Desain Manager: Manager memberikan antarmuka Web kepada administrator maupun penyewa untuk mengelola aplikasi yang disimpan di dalam sistem. Aksi yang dilakukan oleh pengguna dikomunikasikan ke Node maupun Load Balancer. Data terkait penyewa dan aplikasinya seperti daftar pengguna, daftar tagihan (billing), aplikasi Web dan basis data yang tersimpan disimpan dalam basis data berbasis MongoDB.

Manager dibuat dengan konsep *separation of concern* yang terdiri dari Backend berupa REST API dan Frontend berupa aplikasi Web. Penggunaan konsep ini untuk mempermudah pengembangan klien dalam bentuk lain jika diperlukan.

3) Desain Load Balancer: Load Balancer dibuat untuk mendukung tranparansi akses aplikasi Web maupun basis data tanpa perlu mengekspos pada Node mana data aplikasi atau basis data tersebut disimpan. Diagram proses contoh proses akses aplikasi dan basis data oleh klien tertera pada Gambar 3.

Data Node disimpan untuk setiap aplikasi Web maupun basis data yang tersimpan. Jumlah Node di setiap aplikasi dibuat tergantung dari jumlah Node aplikasi Web bersangkutan. Data ini juga disimpan terlebih dahulu disimpan pada Manager untuk menjamin konsistensi pengaturan.

4) Desain Node: Node merupakan tempat dimana aplikasi Web dan basis data sebenarnya ditempatkan. Node tidak terekspos oleh jaringan luar dan hanya bisa diakses melalui



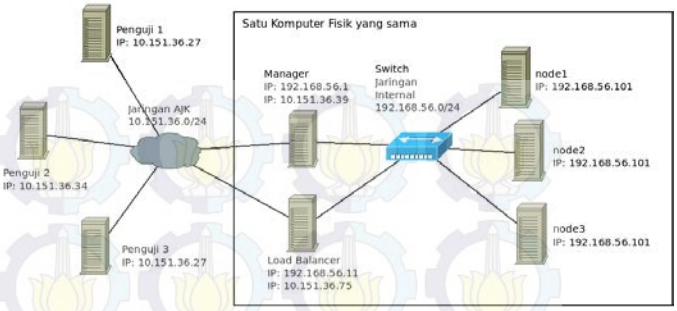
Gambar 3. Diagram Interaksi Proses Akses contoh.com yang Dijalankan di Sistem oleh Pengakses.

jaringan internal melalui Load Balancer. Satu aplikasi Web dapat disimpan pada satu atau beberapa Node dan dapat ditambah/dikurang melalui proses *scaling* oleh pelanggan. Namun untuk basis data (berbasis MySQL), hanya ada satu Node saja untuk satu *server* basis data dikarenakan proses sinkronisasi/replikasi master-master masih belum bisa diaplikasikan secara transparan [10]. Semua data mengenai Node, dan aplikasi Web / basis data yang ter-*hosting* pada suatu Node disimpan pada basis data Hosting Manager.

Jika aplikasi yang sama disimpan dalam beberapa Node yang berbeda pada kasus aplikasi yang menggunakan lebih dari satu Node, akan ada dua jenis klasifikasi Node: sebuah Root Node yang menyipan data master utama aplikasi dan Slave Node yang akan mengikuti data aplikasi dari Root Node. Penyewa dapat mengunggah data aplikasi (berupa kode terek-sekusi) melalui Git ke *root* Node dari aplikasi bersangkutan. Kemudian ketika aplikasi dijalankan, semua Slave Node akan melakukan sinkronisasi data aplikasi dari Root Node.

5) Citra Docker pada Node: Aplikasi atau basis data dijalankan ke dalam sebuah *container* berbasis Docker menggunakan citra yang sesuai dengan platform bahasa pemrograman atau basis data bersangkutan. Pada rancangan tugas akhir ini, terdapat beberapa citra Docker yang dibuat:

- **nodejs-nodemon** citra untuk *container* aplikasi berbasis Node.js. Terdiri dari *server* NGINX sebagai Frontend, Node.js versi 0.10 dan Nodemon (github.com/remy/nodemon) untuk eksekutor aplikasinya.
- **php55-fpm** citra untuk *container* aplikasi berbasis PHP 5.5 menggunakan mesin FastCGI berbasis FPM dan *server* NGINX sebagai Frontend. Terdapat aplikasi composer (http://getcomposer.org) untuk manajemen paket PHP.
- **python27-gunicorn** citra untuk *container* aplikasi berbasis Python versi 2.7 dengan mesin FastCGI berbasis Gunicorn (http://gunicorn.org) dan *server* Frontend NGINX. Container dijalankan dengan virtualenv (http://virtualenv.readthedocs.org) agar pengguna bisa memasang paket Python tambahan yang diperlukan tanpa perlu memanipulasi isi citra.
- **ruby19-thin** citra untuk *container* aplikasi berbasis Ruby 1.9 dengan mesin FastCGI berbasis Thin (code.macournoyer.com/thin/).
- **mysql-single** citra untuk *container* MySQL 5.5. Hanya berjalan satu Node untuk satu *instance* data.



Gambar 4. Toplogi Jaringan dalam Pengujian

6) Lingkungan Implementasi: Lingkungan Implementasi dan pengembangan dilakukan menggunakan komputer PC dengan spesifikasi Intel(R) Core(TM) i3 dengan memori 8GB. Perangkat lunak yang digunakan dalam proses pengembangan antara lain: Sistem operasi Ubuntu Linux 14.04.1 LTS, Desktop xfce, Editor teks vim, git 1.9.1 untuk pengelolaan versi kode program, Node.js 0.10 untuk kerangka kerja pemrograman, Docker 1.2.0 untuk uji coba citra cakram (disk images) Docker, Paket TEXlive untuk penulisan buku tugas akhir dan Peramban *web* Mozilla Firefox.

V. Pengujian dan Evaluasi

A. Lingkungan Uji Coba

Lingkungan untuk pengujian menggunakan lima komputer yang terdiri dari: satu Manager, satu Load Balancer, tiga Node dan tiga komputer penguji. Manager terletak pada komputer sama yang digunakan pada implementasi, sementara Load Balancer dan tiga Node berjalan di bawah VirtualBox versi 4.3 pada komputer tersebut. Selain itu, tiga komputer penguji merupakan komputer fisik di luar komputer yang dilakukan dalam proses implementasi. Gambar 4 menggambarkan topologi jaringan dari masing-masing komputer yang digunakan dalam tahap pengujian.

Proses pengujian dilakukan di Laboratorium Arsitektur dan Jaringan Komputer (IF-307), Lantai tiga Gedung Teknik Informatika, ITS.

Spesifikasi perangkat lunak dan perangkat keras pada masing-masing komputer adalah sebagai berikut:

- Node
  - Perangkat Keras: Virtualisasi VirtualBox 4.3, Prosesor 2x CPU komputer Host, RAM 1024 MB, Hard Disk 8 GB
  - Perangkat Lunak: Ubuntu Linux 14.04 LTS, Node.js 0.10, Docker 1.4.0, Git 1.8, OpenSSH
- Load Balancer
  - Perangkat Keras: Virtualisasi VirtualBox 4.3, Prosesor 2x CPU komputer Host, RAM 1024 MB, Hard Disk 8 GB
  - Perangkat Lunak: Ubuntu Linux 14.04 LTS, Node.js 1.10, HAProxy, OpenSSH

- Penguji
  - Perangkat Keras: Komputer Fisik, Prosesor Intel Pentium, RAM 2 GB, Hard Disk 250 GB
  - Perangkat Lunak: Ubuntu Linux 14.04 LTS, Apache Benchmark, OpenSSH

B. Skenario Uji Coba

Skenario uji coba dilakukan dalam beberapa tahap uji coba:

- **Uji Unit Fungsionalitas** digunakan untuk menguji berjalannya fungsionalitas REST API pada Backend Manager apakah sesuai dengan yang diharapkan.
- **Uji Kapasitas** dilakukan untuk menguji berapa banyak aplikasi dan basis data dari penyewa yang dapat dilayani oleh sistem pada spesifikasi pengujian di atas. Pengujian dilakukan dengan membuat banyak aplikasi dan basis data dengan jenis platform, jumlah *node* yang ditentukan secara acak.
- **Uji Performa** dilakukan untuk menguji bagaimana kecepatan akses dari setiap aplikasi dengan melakukan simulasi akses HTTP. Pengujian dilakukan dengan melakukan *benchmark* pada aplikasi yang dibangun pada uji kapasitas. Uji performa akan melihat berapa persen aplikasi tersedia (bisa diakses dengan sukses) pada 1500 jumlah permintaan dengan 300 koneksi secara bersamaan.

Karena sifat acak yang ada pada uji kapasitas dan keterbatasan jumlah Node yang disediakan, maka pengujian kapasitas serta performa dilakukan sebanyak lima kali untuk bisa melihat lebih lanjut mengenai korelasi antara pemilihan jenis platform, jumlah memori dan jumlah Node dengan performa akses aplikasi bersangkutan.

C. Hasil Uji Coba dan Evaluasi

Berikut dijelaskan mengenai hasil pengujian yang dilakukan pada tiga skenario pengujian yang telah ditentukan.

1) Pengujian Fungsionalitas: Dari hasil uji coba fungsionalitas, didapatkan bahwa Backend pada Manager sudah mengimplementasikan keseluruhan fungsionalitas dengan baik sesuai dengan yang diharapkan. Sebagian besar aksi yang dilakukan pada Backend dapat dilakukan dengan waktu dibawah 100 ms.

Namun, ada beberapa aksi yang memerlukan waktu hingga beberapa detik terutama pada aksi yang membutuhkan pemanggilan skrip panggil pada Node atau Load Balancer seperti pada pembuatan aplikasi/basis data, memulai aplikasi/basis data dan beberapa aksi lainnya. Lamanya akses pada rute ini bergantung pada kecepatan dari masing-masing *node* untuk memproses aksi tersebut.

2) Pengujian Kapasitas dan Performa: Uji kapasitas pada setiap kali percobaan rata-rata berlangsung sekitar 8 - 15 menit. Rata-rata jumlah aplikasi yang dapat tertampung pada setiap percobaan adalah sebanyak 15-20 aplikasi dengan jumlah Node dan penggunaan memori masing-masing aplikasi yang sangat bervariasi.

Secara umum, performa akses pada aplikasi yang memiliki jumlah Node lebih banyak cenderung lebih baik. Beberapa aplikasi bisa mendapatkan indeks ketersediaan hingga 90% - 100%. Namun demikian, faktor platform juga menentukan bagaimana performa aplikasi dapat berjalan. Beberapa platform seperti **php55-fpm** dan **nodejs-nodemon** cenderung memiliki ketersediaan lebih rendah karena pengaturan bawaan dari kedua platform tersebut mungkin belum mengakomodasi sistem aplikasi Web berperforma tinggi yang bisa berjalan pada lingkungan Docker.

Aplikasi yang menggunakan basis data MySQL cenderung memiliki ketersediaan yang lebih rendah dibandingkan aplikasi tanpa basis data. Hal ini memang sesuai eksepektasi karena keterbatasan kemampuan MySQL yang hanya bisa berjalan pada satu Node pada lingkungan sistem untuk saat ini. Namun demikian, ada beberapa aplikasi dengan basis data yang mampu mendapatkan ketersediaan melebihi 60% terutama pada aplikasi yang memiliki jumlah Node lebih dari satu.

D. Implementasi Prinsip Komputasi Awan

1) On-Demand Self-Service: Prinsip ini diimplementasikan pada sistem melalui adanya panel kontrol dari sisi administrator dan penyewa agar mereka dapat mengatur dan mengelola aplikasi dan basis data beserta penggunaan sumber daya yang digunakannya. Panel kontrol di sisi Manager dapat berkomunikasi secara otomatis ke setiap Node dan Load Balancer sesuai permintaan yang ada dari sisi pengguna.

Namun demikian, fitur penagihan atau billing yang ada pada sistem terimplementasi masih bersifat tradisional yang membutuhkan peran serta administrator untuk menyetujui setiap pesanan (walaupun admin tidak perlu melakukan banyak hal untuk menyiapkan sistem) serta peran penyewa memerlukan inisiasi terlebih dahulu ketika aplikasi atau basis data mereka perlu ditingkatkan kemampuannya.

2) Resource-Pooling: Konsep **Resource-Pooling** pada sistem dapat terlihat dari kemampuan sistem untuk menjalankan banyak aplikasi dan basis data pada sejumlah Node yang disediakan. Penggunaan Docker untuk melakukan virtualisasi di setiap aplikasi memungkinkan banyak proses aplikasi berjalan pada sistem dengan menggunakan kernel yang sama, namun dengan tetap menjaga transparansi sistem antara satu sama lain. Gambar 5 memperlihatkan banyaknya proses di dalam Docker yang untuk semua aplikasi dan basis data pada sebuah Node.

3) Measured Service: Sifat layanan terukur pada sistem dapat dilihat dari adanya penekanan pada pengaturan kapasitas jumlah Node, jumlah memori dan jumlah penggunakan cakram penyimpanan pada setiap aplikasi atau basis data yang ada pada penyewa. Sistem memiliki fitur untuk melakukan pemantauan terhadap penggunaan ketiga komponen tersebut serta fitur untuk memantau penggunaan aplikasi Web itu sendiri dalam bentuk laporan (seperti pada gambar 6) yang dapat memberikan pengetahuan kepada pelanggan mengenai bagaimana aplikasi mereka diakses oleh klien.

VI. PENUTUP

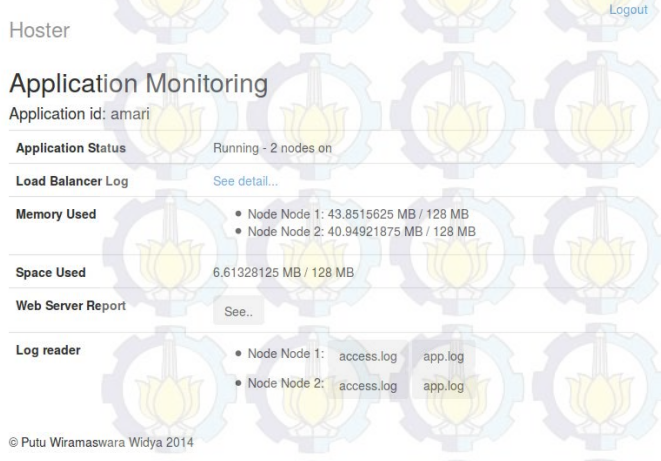
A. Kesimpulan

Dari proses perancangan, implementasi dan pengujian yang dilakukan terhadap sistem, dapat diambil beberapa kesimpulan sebagai berikut :

```
root@node1:/opt/tugasakhir# docker ps
CONTAINER ID        IMAGE                              COMMAND             CREATE
D                   STATUS              PORTS
 NAMES
e829cc125586        hoster/python27-gunicorn:latest    "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50827->22/tcp, 0.0.0.0:50828->80/tcp
 app5496bbcdd90da29a78b7c1c9
f5dacbe68547        hoster/python27-gunicorn:latest    "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50825->22/tcp, 0.0.0.0:50826->80/tcp
 app5496bbb9d90da29a78b7c1c7
a41ae748f46d        hoster/mysql-single:latest         "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50824->3306/tcp
 db5496bbbcd90da29a78b7c1c8
d9cb70c602a3        hoster/python27-gunicorn:latest    "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50822->22/tcp, 0.0.0.0:50823->80/tcp
 app5496bba2d90da29a78b7c1c5
2cbb126ffe85        hoster/mysql-single:latest         "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50821->3306/tcp
 db5496bba4d90da29a78b7c1c6
1b35535d0387        hoster/ruby19-thin:latest          "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50820->80/tcp
 app5496bb98d90da29a78b7c1c4
bd4731139876        hoster/ruby19-thin:latest          "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50819->80/tcp
 app5496bb7fd90da29a78b7c1c2
ada705ce7c93        hoster/mysql-single:latest         "/sbin/my_init"     3 hour
s ago               Up 3 hours          0.0.0.0:50818->3306/tcp
 db5496bb80d90da29a78b7c1c3
root@node1:/opt/tugasakhir#
```

Gambar 5. Hasil dari Perintah `docker ps` yang Menampilkan Daftar Kontainer Docker yang Berjalan untuk Mengakomodir Banyaknya Aplikasi dan Basis Data pada Suatu Node

1) Sistem Platform-as-a-Service untuk kebutuhan pengembangan aplikasi dan Web *hosting* berbasis *multi*-tenancy dapat diimplementasikan menggunakan berbagai kombinasi kerangka kerja yaitu pemrograman berbasis Node.js dan virtualisasi berbasis Docker. Sistem telah dikembangkan sesuai fitur yang dirancang meliputi dukungan *multi*-platform, dukungan untuk peningkatan atau *scaling* sumber daya komputasi serta fitur pemantauan terhadap berjalannya aplikasi atau basis data dari masing-masing penyewa atau *tenant*. Fitur tersebut telah teruji menggunakan ujit unit fungsionalitas serta uji kapasitas dan performa.
2) Rata-rata jumlah aplikasi yang dapat tertampung pada konfigurasi tiga Node dengan masing-masing memori 1 GB adalah sekitar 12 - 18 aplikasi tergantung dari konfigurasi aplikasi. Secara umum, aplikasi dengan jumlah Node lebih banyak memiliki ketersediaan paling tinggi hingga 100%, sementara aplikasi dengan basis data memiliki kecenderungan ketersediaan yang lebih rendah daripada aplikasi yang tidak terkoneksi basis data sama sekali.
3) Prinsip komputasi awan yaitu *self-service*, *resource-pooling* dan *measured service* dapat diimplementasikan dengan baik pada sistem seperti dijelaskan pada poin sebelumnya.



Gambar 6. Antarmuka Pemantauan Sebuah Aplikasi Secara Umum, Dapat Dilihat Penggunaan Memori dan Cakram Penyimpanan pada Berjalannya Aplikasi Bersangkutan

B. Saran

Berikut adalah beberapa saran-saran yang diberikan untuk pengembangan lebih lanjut :

- Perlu adanya penelitian lebih lanjut dari sisi ekonomi mengenai adanya layanan semacam ini apakah dapat diaplikasikan dalam dunia nyata.
- Penggunaan Docker untuk menjalankan proses *server* Web dan basis data perlu diteliti lebih lanjut dalam hal konfigurasi yang cocok sehingga bisa menjalankan aplikasi Web pengguna dengan efisien dan handal.
- Perlu adanya penelitian lebih lanjut mengenai penggunaan basis data dengan sistem *multi-node* sehingga bisa memaksimalkan luaran kecepatan akses basis data pada sistem ini.
- Perlu adanya peningkatan pada sistem penagihan secara otomatis misalnya dengan mekanisme poin dan kredit sehingga proses yang membutuhkan otorisasi biaya seperti *scaling* dan pembuatan aplikasi baru dapat dilakukan secara otomatis tanpa memerlukan persetujuan dari administrator.

PUSTAKA

[1] Gregory Go, **7 types of Web Hosting**, [Online], http://onlinebusiness.about.com/od/webhosting/tp/web-hosting-types.htm, diakses tanggal 18 September 2014
[2] Yashpalsinh Jadeja, Kirit Modi, **Cloud Computing - Concepts, Architecture and Challenges**, 2012 International Confrence on Computing, Electronics and Electrical Technologies [ICCEET], published by IEEE, pp 887-880, 2012
[3] National Institute of Standards and Technology, **The NIST Definition of Cloud Computing**, [Online], http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, diakses tanggal 22 September 2014.
[4] Guillermo Rauch, **Smashing Node.js: JavaScript Everywhere**, Wiley, First Edition, 2012
[5] Joyent, inc. ***node*.js**, [Online], http://www.nodejs.org/, Diakses tanggal 2 Desember 2014
[6] Stefan Tilkov, Steve Vinoski, **Node.js: Using JavaScript to Build High-Perfomace Network Programs**, IEEE Computer Society Issue 06, Nov-Dec 2010, pp 80-83, 2010
[7] Scott Davis, **Mastering MEAN: Introducing to MEAN Stack**, [Online], http://www.ibm.com/developerworks/library/wa-mean1/index.html, diakses tanggal 21 Desember 2014
[8] James Turnbull, **The Docker Book: Containerization is a the new Virtualization**, version 1.3.2, James Turnbull, 2014
[9] HAProxy, **HAProxy Description**, [Online], http://www.haproxy.org/#desc, diakses tanggal 15 Desember 2014.
[10] Adam Wigins, Heroku **SQL Database Don't Scale**, [Online], http://adam.herokuapp.com/past/2009/7/6/sql_databases_dont_scale/, diakses tanggal 18 September 2014.