TUGAS AKHIR - KI141502

RANCANG BANGUN LAYANAN PLATFORM AS A SERVICE (PAAS) UNTUK MENDUKUNG SISTEM MULTI-TENANCY PENGEMBANGAN APLIKASI BERBASIS KOMPUTASI AWAN

PUTU WIRAMASWARA WIDYA
NRP 5111 100 012

Dosen Pembimbing 1
Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D.

Dosen Pembimbing 2
Baskoro Adi Pratomo, S.Kom, M.Kom

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2015

UNDERGRADUATE THESES - KI141502

DESIGN AND IMPLEMENTATION OF PLATFORM AS A SERVICE TO SERVE A MULTI-TENANCY APPLICATION PLATFORM BASED ON CLOUD COMPUTING

PUTU WIRAMASWARA WIDYA
NRP 5111 100 012

Supervisor 1
Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D.

Supervisor 2
Baskoro Adi Pratomo, S.Kom, M.Kom

INFORMATICS DEPARTMENT
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2015

RANCANG BANGUN LAYANAN PLATFORM AS A SERVICE (PAAS) UNTUK MENDUKUNG SISTEM MULTI-TENANCY PENGEMBANGAN APLIKASI BERBASIS KOMPUTASI AWAN

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
Putu Wiramaswara Widya
NRP: 5111 100 012

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D. .....
NIP: 197708242006041001 (Pembimbing 1)

Baskoro Adi Pratomo, S.Kom, M.Kom .....
NIP: 198702182014041001 (Pembimbing 2)

SURABAYA
JANUARI 2015

RANCANG BANGUN LAYANAN PLATFORM AS A SERVICE (PAAS) UNTUK MENDUKUNG SISTEM MULTI-TENANCY PENGEMBANGAN APLIKASI BERBASIS KOMPUTASI AWAN

Nama	: PUTU WIRAMASWARA WIDYA
NRP	: 5111 100 012
Jurusan	: Teknik Informatika FTIf-ITS
Pembimbing I	: Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D.
Pembimbing II	: Baskoro Adi Pratomo, S.Kom, M.Kom

Abstract *Layanan hosting aplikasi atau sering disebut Application Hosting / Web Hosting merupakan layanan yang jamak tersedia di Indonesia. Layanan ini memberikan wadah bagi siapapun untuk menempatakan aplikasi Web mereka beserta basis datanya di jaringan Internet tanpa perlu menyediakan sendiri infrastruktur Internet terdedikasi. Layanan ini memiliki berbagai macam bentuk. Salah satu yang paling banyak tersedia di Indonesia adalah Shared Web Hosting yang menyediakan layanan yang sangat murah. Pada jenis layanan ini, penyewa hanya diberikan satu buah platform yang sama dengan aplikasi yang disimpan bersamaan pada satu buah server sehingga memungkinkan adanya bottleneck. Selain itu, terdapat jenis layanan lain yaitu Virtual Private Server (VPS) yang menyediakan kendali penuh pada sistem melalui teknik yang disebut virtualisasi namun dengan konfigurasi yang sangat kompleks.*

Saat ini terdapat jenis layanan hosting baru yang disebut dengan 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 menawarkan fitur untuk transparansi akses, multi-platform, skalabilitas, reliabilitas, keterbukaan API dan kemudahan pengguna. Namun, layanan ini saat masih sangat jarang terutama di Indonesia yang masih menggunakan platform yang sama dengan Shared Web Hosting.

Pada tugas akhir ini, dilakukan perancangan dan implementasi sistem hosting atau pengembangan aplikasi yang mendukung banyak tenant/penyewa (Multi-Tenancy), mendukung banyak platform dan mendukung tiga prinsip komputasi awan yaitu self-service, resource pooling dan metered service. Sistem ini mengadopsi beberapa fitur yang ada di Cloud Web Hosting modern sehingga bisa dibangun oleh pihak ketiga dengan mudah.

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

DESIGN AND IMPLEMENTATION OF PLATFORM AS A SERVICE TO SERVE A MULTI-TENANCY APPLICATION PLATFORM BASED ON CLOUD COMPUTING

Name : PUTU WIRAMASWARA WIDYA
NRP : 5111 100 012
Major : Informatics Department Faculty of IT - ITS
Supervisor I : Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D.
Supervisor II : Baskoro Adi Pratomo, S.Kom, M.Kom

Abstract

Application hosting services or usually known as web hosting is one of the most commonly available Internet services in Indonesia. This services provide anyone an ability to host their own application and its database. There are many forms of application hosting service. The one that abudantly available in Indonesia is shared web hosting that provide a single homogenous platform to serve a numerous number of web application or database inside a single machine. The bottom line is that this services have a potential bottleneck issue due to its shared nature. In addition, there is another form of web hosting which provides its tenant an ability to take control over its operating system yet it has a complex configuration which known as Virtual Private Server.

The third form of web hosting, which gaining its momentum recently, is cloud-based web hosting. It adopts many characteristics and principles from cloud computing technology, that mainly based on Platform-as-a-Service service model. Furthermore, it provides some interesting features such as access transparancy, multiplatform, scalability, reliability, open API access and usability in terms of its interface. However, the service providers of this model is scarce. Even in Indonesia itself, there are few hosting services which labeled as "cloud hosting service" despite still continuing to use shared web hosting-based platform.

This final project/theses assignment aims to build a small appli-

cation development and hosting Platform-as-a-service with multi-platform ability and some other features to fullfil three characteristics of cloud computing: self-service, resource pooling and metered service. The objective is to create an easy to build and cheap application hosting service with its feature based from the limitation of traditional cPanel based service and common feature available in the modern cloud hosting.

Keywords: Application Hosting, Platform-as-a-Service, Multi-Tenancy, Cloud Computing

KATA PENGANTAR

Om Swastyastu

Puji syukur penulis haturkan kepada Ida Sang Hyang Widhi Wasa, Tuhan Yang Maha Esa karena atas *asungkertha wara nugraha* beliau, penulis dapat menyelesaikan sebuah penugasan Tugas Akhir untuk gelar Sarjana Komputer penulis dengan judul **"Rancang Bangun Layanan Platform as a Service (PAAS) untuk Mendukung Sistem Multi-Tenancy Pengembangan Aplikasi Berbasis Komputasi Awan"**.

Tugas Akhir ini dibuat berdasarkan pengalaman penulis dalam melakukan pengembangan aplikasi Web selama masa kuliah menggunakan Node.js. Sampai saat ini, masih belum ada layanan Web Hosting murah yang menawarkan segala fleksibilitas komputasi awan seperti layaknya layanan Heroku atau OpenShift. Diharapkan dengan terselesaikannya Tugas Akhir ini, akan ada peluang yang terbuka bagi pengusaha di Indonesia untuk mengembangkan layanan jenis ini sehingga pengembangan aplikasi Web bisa lebih heterogen dengan berbagai adanya dukungan banyak platform.

Tidak lupa penulis sampaikan terima kasih dengan segala rasa penghormataan seluas-luasnya kepada beberapa pihak yang telah membantu dalam menyelesaikan Tugas Akhir ini selama sekitar tiga bulan :

- Kedua orang tua penulis, saudara dan kerabat dekat yang selalu khawatir dengan keadaan penulis dan selalu mendoakan kelancaran pengerjaan Tugas Akhir ini.
- Bapak Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D. selaku pembimbing I yang selalu menuntun dan membimbing penulis dalam pelaksanaan konsep pengerjaan Tugas Akhir ini.
- Bapak Baskoro Adi Pratomo, S.Kom, M.Kom selalu pembimbing II yang berkenan memberikan evaluasi terhadap perkembangan pengerjaan Tugas Akhir ini.
- Ibu Dr. Eng. Nanik Suciati, S.Kom, M.Kom selaku Ketua Ju-

rusan Teknik Informatika ITS yang telah memberi teladan baik kepada mahasiswa-mahasiswi beliau se-Jurusan.

- Ibu Isye Arieshanti, S.Kom, M.Phil, selaku Ketua Program Studi S1 Teknik Informatika yang telah meluangkan segenap waktunya untuk mengatur berjalannya perkuliahan di jurusan Teknik Informatika ITS.
- Bapak Radityo Anggoro, S.Kom, M.Sc selaku koordinator Tugas Akhir yang telah meluangkan waktunya untuk kelancaran pelaksanaan Tugas Akhir di jurusan.
- Ibu Prof. Handayani Tjandrasa selaku dosen wali dan pembibing akademik penulis selama menempuh masa perkuliahan.
- Teman-teman Administrator di Laboratorium Arsitektur dan Jaringan Komputer (AJK) yang telah memberi suasana yang hangat dan akrab selama pengerjaan TA yaitu Samihd, Uyung, Vivi, Harum, Surya, Agus, Adi Pur, Evaria, Romen, Nisa, Saiful, Wicak dan Zaza.
- Seluruh staf pengajar dan administrasi jurusan Teknik Informatika ITS yang telah memberikan segala tenaganya untuk memajukan perkembangan Jurusan.
- Teman-teman satu jurusan Teknik Informatika ITS angkatan 2010, 2011, 2012 dan 2013 yang tidak bisa penulis sebutkan satu persatu namanya.
- Teman-teman perkumpulan Kelompok Linux arek Suroboyo (KLAS) yang selalu mengadakan cangkruan di setiap bulannya.
- Teman-teman komunitas BlankOn Linux di Internet yang sudah memberi banyak ilmu baru mengenai pengembangan Linux dan teknologi jaringan.
- Serta berbagai pihak yang tidak bisa penulis sebutkan satu per satu namanya.

Penulis menyadari bahwasanya Tugas Akhir ini masih jauh dari sempurna. Penulis mengharapkan saran dan kritik yang membangun untuk lebih menyempurnakan penelitian ini kedepannya.

Terima Kasih.
Om Santhi Santhi Santhi Om

Surabaya, 22 Desember 2014
Putu Wiramaswara Widya

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Layanan *hosting* dan pengembangan aplikasi dalam bentuk Web atau sering disebut dengan Web *hosting* merupakan layanan yang jamak tersedia di Indonesia. Layanan ini memberikan wadah bagi individu, organisasi atau perusahaan besar agar bisa meletakkan aplikasi buatan mereka beserta basis datanya di Internet tanpa perlu dipersulit dengan penyediaan koneksi Internet terdedikasi. Umumnya, aplikasi yang diletakkan berupa situs halaman portal, Web blog, sistem informasi skala kecil hingga besar.

Web Hosting memiliki berbagai macam jenis berdasarkan layanan yang diberikan [1], kemudahan yang disediakan serta teknik penempatan (*host*) aplikasinya. 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 dan setiap aplikasi bisa diakses melalui nama domain yang ditentukan. Konsekuensi dari penyimpanan pada *server* yang sama adalah adanya kemungkinan *bottleneck* baik dari sisi pemrosesan, memori dan jaringan. Selain itu, Shared Web Hosting hanya menyediakan satu platform yang sama 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* karena VPS memberikan kendali sistem operasi secara penuh melalui teknik yang disebut dengan virtualisasi. VPS tersedia dalam berbagai macam jenis layanan berdasarkan jumlah sumber daya penyimpanan memori, jaringan serta kemampuan pemrosesan. Selain itu, kebanyakan layanan penyedia VPS saat ini juga memberikan kemampuan peningkatan kapasitas dan fitur pemantauan. 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 (dijelaskan pada tabel A.1 pada lampiran) 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*) (perbandingan fitur antara *hosting* cPanel dengan Cloud Web Hosting lainnya dijelaskan pada tabel A.2 pada lampiran). 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 [9] 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.

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah dalam Tugas Akhir ini:

- Bagaimana membangun sistem Platform-as-a-Service (PaaS) yang dirancang untuk penyedia jasa layanan *hosting* aplikasi?
- Bagaimana prinsip on-demand *self-service*, *resource pooling* dan *measured service* dalam komputasi awan diterapkan di dalam perangkat lunak PaaS yang dirancang?

1.3 Batasan Masalah

Batasan masalah pada Tugas Akhir ini adalah sebagai berikut:

- Sistem yang dibangun mengadopsi sifat komputasi awan dibatasi pada: *self-service*, *resource pooling* dan *measured service*.
- Sistem dibatasi untuk kebutuhan *hosting* aplikasi, eksekusi aplikasi pada Web (HTTP) serta basis data.
- Platform bahasa pemrograman yang didukung dibatasi pada: Node.js, PHP, Python dan Ruby. Sementara basis data dibatasi pada MySQL.
- Modul pembagi muat hanya akan diberlakukan pada aplikasi Web. Sementara basis data hanya dapat berjalan pada satu *node* saja.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah menghasilkan sistem dan perangkat lunak Platform-as-a-Service untuk kebutuhan pengembangan dan *hosting* aplikasi pada Web yang mendukung beberapa sifat komputasi awan.

1.5 Manfaat

Manfaat yang diberikan dengan pembuatan Tugas Akhir ini adalah memberikan peluang kepada pihak penyedia jasa layanan *web hosting* agar dapat menggunakan perangkat luaran untuk membangun layanan *web hosting* yang berbasis komputasi awan.

1.6 Metodologi

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

- **Penyusunan Proposal Tugas Akhir**
  Penyusunan proposal Tugas Akhir dilaksanakan untuk merumuskan masalah serta melakukan penetapan desain dasar sistem yang akan dikembangkan dalam pelaksanaan Tugas Akhir ini.
- **Studi Literatur**
  Untuk membantu proses pengerjaan Tugas Akhir, diperlukan studi lebih lanjut mengenai penggunaan komponen-komponen terkait dengan sistem yang akan dibangun.
- **Desain dan Perancangan**
  Dalam rangka memerinci lebih jauh mengenai bagaimana memanfaatkan komponen-komponen sistem untuk membangun sistem secara utuh, diperlukan proses desain dan perancangan dari sistem. Hasil analisis dan desain kemudian ditetapkan menjadi rancangan dasar implementasi sistem.
- **Implementasi Sistem**
  Hasil analisis dan desain kemudian diimplementasikan melalui komponen-komponen perangkat lunak pendukung.
- **Uji Coba dan Evaluasi**
  Untuk mengukur kemampuan sistem untuk menangani pengguna dari segi fungsionalitas dan performa, dilakukan proses uji coba dan evaluasi untuk mengetahui sejauh mana sistem dapat berjalan sesuai dengan yang diharapkan dan sejauh mana prinsip dari komputasi awan dapat diimplementasikan di dalam sistem.

1.7 Sistematika Laporan

Buku Tugas Akhir ini terdiri dari beberapa bab yang dijelaskan sebagai berikut:

- **Bab 1. Pendahuluan**. Bab ini berisikan latar belakang, rumusan masalah, tujuan, manfaat, metodologi dan sistematika

penulisan yang digunakan sebagai dasar penyusunan Tugas Akhir.

- **Bab 2. Tinjauan Pustaka**. Bab ini berisikan teori penunjang yang berhubungan dengan Tugas Akhir.
- **Bab 3. Desain dan Perancangan**. Bab ini membahas desain dasar dari sistem dan perangkat lunak yang akan dirancang sebagai bagian dari pengerjaan Tugas Akhir.
- **Bab 4. Implementasi**. Bab ini membahas hasil implementasi dari rancangan sistem beserta tekniknya.
- **Bab 5. Uji Coba dan Evaluasi**. Bab ini membahas mengenai teknik uji coba dan hasil keluaransnya sebagai bahan evaluasi terhadap hasil Tugas Akhir.
- **Bab 6. Penutup**. Bab ini berisi kesimpulan dari hasil uji coba serta saran untuk pengembangan Tugas Akhir selanjutnya.

Halaman ini sengaja dikosongkan

BAB 2

LANDASAN TEORI

2.1 Web Hosting

Web Hosting merupakan jasa penyewaan ruang dan koneksi untuk keperluan menempatkan suatu aplikasi atau halaman Web di Internet. Layanan ini memberikan kesempatan kepada pihak ketiga yang tidak memiliki koneksi Internet terdedikasi (dengan IP publik) untuk menempatkan situs Web mereka melalui *server* yang disediakan oleh pihak Web Hosting.

Web Hosting memiliki beberapa jenis berdasarkan metode penempatan berkas Web dan fasilitas yang diberikannya [1]:

- **Shared Hosting**, merupakan jenis *web hosting* yang paling umum. Pada *shared hosting*, situs Web setiap pelanggan diletakkan secara bersamaan pada satu *server* sehingga semua pelanggannya hanya perlu menanggung operasional satu *server* tersebut. Kekurangan utama dari *shared hosting* adalah adanya perebutan sumber daya yang dilakukan jika satu situs Web diakses lebih banyak daripada yang lainnya. Namun kelebihannya, harga yang ditawarkan jauh lebih murah dibandingkan *hosting* lainnya.
- **Reseller Hosting**, merupakan perangkat *shared web hosting* yang bisa dijual ke orang lain yang membutuhkan.
- **Grid/Cloud Hosting**, merupakan jenis *web hosting* baru yang memungkinkan penggunaan beberapa *server* untuk digabung menjadi sebuah *server* besar. Penggunaannya adalah ketika kemampuan sumber daya yang dibutuhkan lebih banyak, maka hanya perlu penambahan sumber daya dan komputer untuk bergabung ke jaringan sebelumnya. Umumnya, cloud *hosting* menggunakan skema pasca-bayar yaitu membayar sesuai dengan penggunaan sumber daya.
- **Virtual Private Server (VPS)** merupakan jenis *hosting* yang memberikan fasilitas pengendalian dan akses sistem operasi *server* secara penuh. Walaupun VPS menggunakan teknik

virtualisasi untuk membagi sumber daya satu *server*, namun pembagian sumber dayanya terjamin sehingga tidak terjadi rebutan sumber daya akibat ketimpangan akses.

- **Dedicated Server** merupakan jenis *hosting* yang memberikan satu unit *server* utuh secara fisik dan memberikan akses penuh kepada pelanggannya.
- **Colocation** merupakan jenis *hosting* yang hanya memberikan perangkat dan ruang bagi pelanggannya untuk membuat pusat data sendiri. Pelanggan diwajibkan menggunakan komputer mereka sendiri dan layanan hanya menyediakan listrik, pendinginan, keamanan fisik, dan koneksi Internet.
- **Self Service** merupakan layanan *hosting* yang disediakan sendiri oleh pelanggannya tanpa memerlukan layanan *web hosting* lain. Pelanggan hanya membutuhkan koneksi Internet terdedikasi dan bertanggungjawab untuk segala kebutuhan sumber daya lainnya.

2.2 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. [9]

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. [8]

Tujuan utama dari komputasi awan adalah untuk membuat penggunaan sumber daya terdistribusi menjadi lebih baik dengan meng-

ombinasikannya 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.

2.2.1 Karakteristik

Berikut adalah beberapa karakteristik umum yang terdapat pada komputasi awan menurut definisi dari NIST [9]:

- *On-Demand Self Service*, 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)

2.2.2 Model Layanan

Secara umum, terdapat tiga model layanan yang umumnya ditawarkan pada komputasi awan [9]:

- *Infrastructure as a Service (IaaS)* memberikan layanan kepada pelanggan untuk menetapkan penyimpanan, jaringan dan sumber daya komputer utama lainnya dimana pelanggan da-

pat menjalankan perangkat lunak apa saja didalamnya dan melakukan kendali terhadap sistem operasi, penyimpanan dan aplikasi yang disebarkannya.

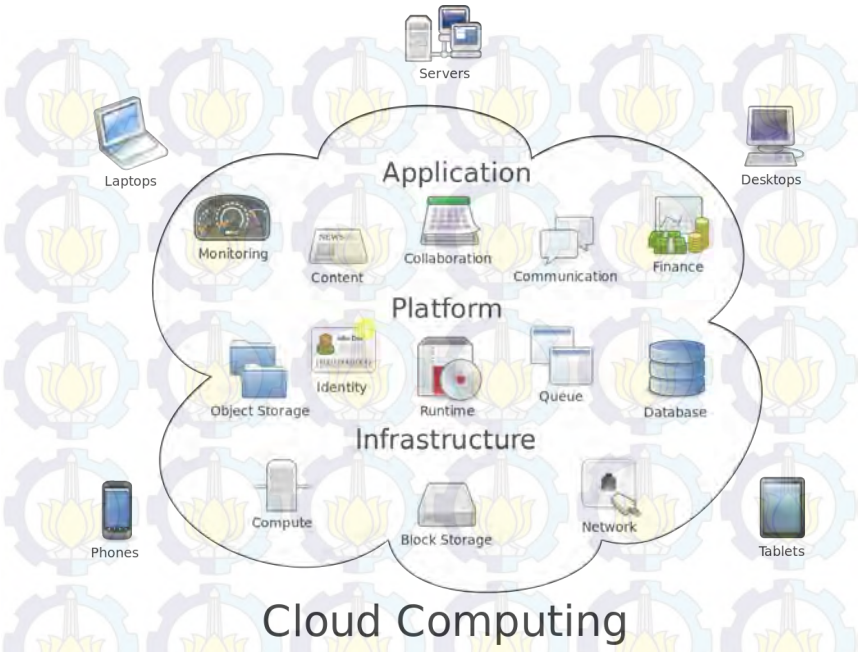
- *Platform as a Service (PaaS)* memberikan layanan kepada pelanggan untuk menyebarkan (*deploy*) aplikasi kepada pengguna lainnya melalui bahasa pemrograman, pustaka, layanan dan kakas yang didukung oleh penyedia. Pelanggan tidak perlu mengontrol infrastruktur di bawahnya seperti jaringan, *server*, sistem operasi dan media penyimpanan namun memiliki kendali terhadap aplikasi yang mereka kembangkan.
- *Software as a Service (SaaS)* memberikan layanan kepada pelanggan untuk menggunakan perangkat lunak aplikasi pada infrastruktur komputasi awan. Aplikasi bisa diakses dari berbagai macam klien, baik melalui antarmuka *thick-client* seperti aplikasi desktop maupun *thin-client* seperti peramban Web.

Ketiga layanan tersebut digambarkan pada Gambar 2.1

2.3 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. [10]

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 aplika-



Gambar 2.1: Ilustrasi Tiga Model Layanan Komputasi Awan

si yang bersifat waktu nyata dan intensif data. [11] Contoh kode program untuk membuat *server* Web sederhana bisa dilihat pada gambar 2.2

Node.js mengadopsi konsep pemrograman berbasis *event* dibandingkan menggunakan thread untuk mengelola banyaknya masukan dari I/O seperti pada aplikasi jaringan yang menerima banyak klien. Untuk menggunakan *event*, pemrogram menuliskan kode yang akan dieksekusi ketika suatu *event* terjadi misalnya ketika ada permintaan koneksi dari klien atau ketika data dari basis data siap untuk dikirimkan ke klien. Ketika *event* terjadi, sistem notifikasi

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

Gambar 2.2: Contoh Kode Server Web Sederhana Menggunakan Node.js

akan memberitahu kepada aplikasi untuk mengelola *event* tersebut.

Konsep *event* berlanjut pada penggunaan teknik yang disebut Asynchronous I/O. Pada teknik ini, program tidak akan terblokir atau dihentikan sementara jika melakukan suatu operasi I/O, melainkan program akan tetap berlanjut. Ketika operasi I/O berhasil dilakukan, program akan kembali melakukan operasi semestinya terhadap I/O tersebut. Hal ini sangat berguna pada aplikasi yang sangat berat dalam melakukan proses I/O seperti operasi basis data atau berkas. [12]

2.4 MEAN Framework

MEAN Framework atau MEAN Stack merupakan kerangka kerja pemrograman yang terdiri dari komponen berikut [13]:

- 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.

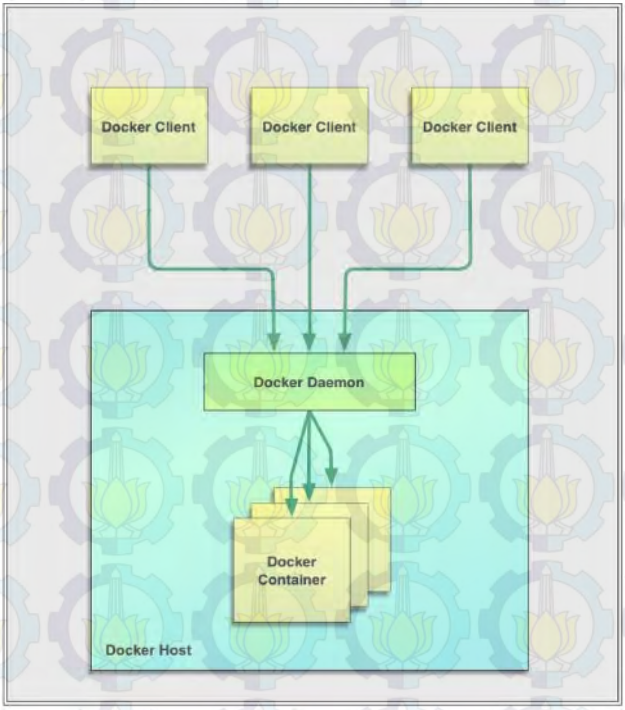
2.5 Docker

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`. [14] Selain itu, pengguna juga dapat membangun citra mereka sendiri menggunakan API yang sudah disediakan.

Docker berjalan dengan konsep klien dan *server*. Pada sisi *server*, berjalan sebuah Docker Daemon yang akan mengelola jalan-

nya setiap *container* yang ada di dalam *server*. Docker menggunakan sistem berkas AUFS untuk mengelola berkas pada setiap *container* dan menggunakan LXC untuk mengatur jalannya *container*. Arsitektur dasar dari Docker dapat dilihat pada Gambar 2.3



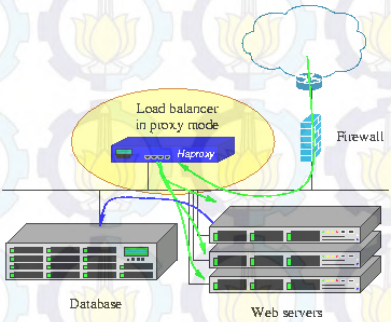
Gambar 2.3: Arsitektur Dasar dari Docker

Docker dan virtualisasi tingkat sistem operasi pada umumnya merupakan salah satu komponen utama dari suatu sistem PaaS karena adanya kebutuhan khusus pada aplikasi yang berjalan di bawah PaaS seperti: isolasi aplikasi, pengurangan *overhead* dari performa jika dibandingkan dengan mesin *virtual hypervisor*, dukungan berbagai platform dan bahasa pemrograman, dan perpindahan posisi

aplikasi dari satu mesin dan mesin lain dengan mudah. [15]

2.6 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 [16]. Ilustrasi penggunaan HAProxy dapat dilihat pada Gambar 2.4.



Gambar 2.4: Ilustrasi Penggunaan HAProxy sebagai Penyeimbang Muat di Belakang Firewall

Halaman ini sengaja dikosongkan

BAB 3

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis, perancangan dan implementasi dari sistem.

3.1 Kasus Penggunaan

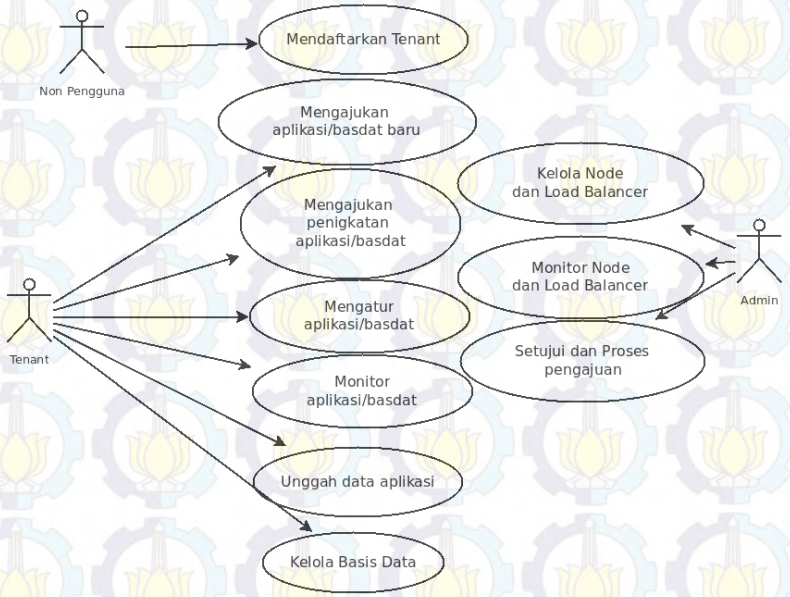
Gambar 3.1 menampilkan kasus penggunaan sistem secara umum dengan penjelasan tertera pada Tabel 3.1

Tabel 3.1: Daftar Kasus Penggunaan Sistem

No	Nama	Aktor	Deskripsi
UC01	Mendaftarkan Penyewa	Non Pengguna	Calon penyewa yang berminat untuk menggunakan sistem diwajibkan utnuk mendaftarkan sebuah akun ke sistem terlebih dahulu
UC02	Mengajukan Aplikasi/Basis Data Baru	Penyewa	Aplikasi dan basis data yang ingin dibuat terlebih dahulu diajukan oleh penyewa. Kemudian penyewa akan diberi kesempatan untuk melakukan pembayaran atas tagihan dari pengajuan tersebut dan Admin menyetujuinya.
UC03	Mengajukan Peningkatan (*scaling*) Aplikasi/Basis Data	Penyewa	Jika aplikasi dan basis data ingin ditingkatkan kapasitasnya oleh penyewa, maka penyewa perlu melakukan pengajuan peningkatan tersebut.

No	Nama	Aktor	Deskripsi
UC04	Pengaturan Aplikasi/Basis Data	Penyewa	Aplikasi dan basis data yang sudah berjalan dapat diatur oleh pengguna. Pengaturan meliputi: nama domain dan jalan tidaknya instance aplikasi.
UC05	Monitor Aplikasi/Basis Data	Penyewa	Aplikasi dan basis data yang sudah berjalan dapat dimonitor oleh pengguna. Monitor meliputi penggunaan sumber daya diska, memori, rangkuman log akses dan rangkuman log sistem secara umum.
UC06	Unggah Data Aplikasi	Penyewa	Data berkas aplikasi dapat diunggah oleh penyewa pemilik aplikasi melalui antarmuka berbasis Git.
UC07	Kelola Basis Data	Penyewa	Basis data dapat dimanipulasi melalui akses konsol dari aplikasi basis data
UC08	Kelola Node dan Load Balancer	Admin	Semua data Node dan Load Balancer yang ada di sistem dikelola oleh admin sistem.
UC09	Setujui dan Proses Penagihan	Admin	Pengajuan dan penambahan (*scaling*) aplikasi dan basis data disetujui oleh admin untuk kemudian diaktivasi.

No	Nama	Aktor	Deskripsi
UC10	Monitor Node dan Load Balancer	Admin	Monitor Node dan Load Balancer yang terdaftar secara umum seperti penggunaan diska, memori dan pembacaan berkas log penting



Gambar 3.1: Diagram Kasus Penggunaan Sistem

3.2 Deskripsi Fitur

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

3.2.1 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.

3.2.2 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.

3.3 Arsitektur Sistem

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

3.3.1 Desain Umum Sistem

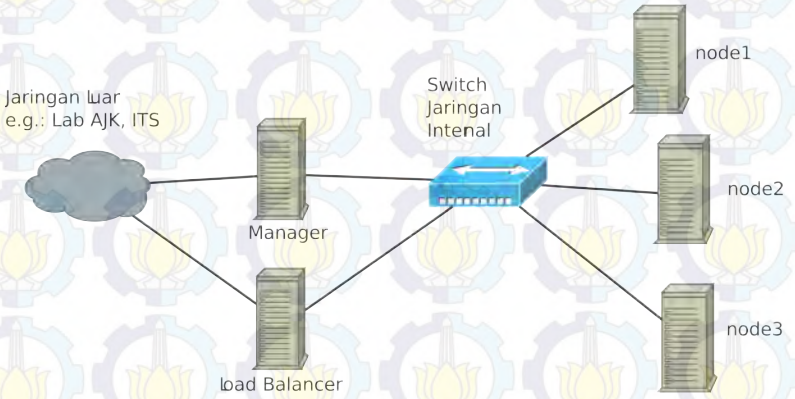
Sistem dibangun dalam beberapa komponen umum yang terkait satu sama lain, yaitu:

- **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 3.2 menampilkan diagram arsitektur hubungan antara komponen.



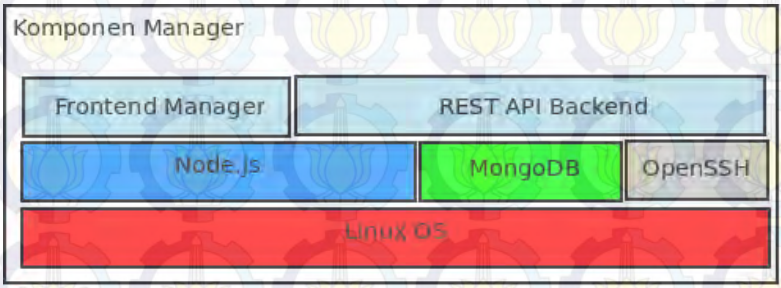
Gambar 3.2: Desain Sistem Secara Umum

3.3.2 Desain Rinci 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 Mo-

ngoDB.

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. Diagram arsitektur Manager tertera pada Gambar 3.3.



Gambar 3.3: Desain Arsitektur pada Manager

3.3.2.1 Backend Manager

Bagian Backend Manager dibuat menggunakan platform Node.js dan Express.jsi pada sisi *server*. Tabel 3.2 menjelaskan daftar rute akses REST API yang ada pada Backend. Ada tiga jenis rute pada API ini: rute yang bisa diakses tanpa autentikasi, rute yang bisa diakses oleh penyewa dan rute yang hanya bisa diakses oleh admin.

Tabel 3.2: Daftar Rute REST API pada Backend

No	Rute	Metode	Hak Akses	Aksi
1	/auth/request	POST	Tidak ada	Mendapatkan token berdasarkan kredensial pengguna

No	Rute	Metode	Hak Akses	Aksi
2	/auth/verify	POST	Tidak ada	Melakukan verifikasi keabsahan token
3	/users	GET, POST, PUT, DELETE	Admin	Melakukan manipulasi data pengguna penyewa
4	/apps	GET, POST, PUT, DELETE	Admin	Melakukan manipulasi data aplikasi
5	/apps/*app-id*/operation	POST	Penyewa	Melakukan suatu operasi pada aplikasi tertentu
6	/dbs	GET, POST, PUT, DELETE	Admin	Melakukan manipulasi data basis data
7	/dbs/*db-id*/operation	POST	Penyewa	Melakukan suatu operasi pada basis data tertentu
8	/nodes	GET, POST, PUT, DELETE	Admin	Melakukan manipulasi data Node dan Load Balancer
9	/nodes/node-*id*/operation	POST	Admn	Melakukan suatu operasi pada *node* tertentu

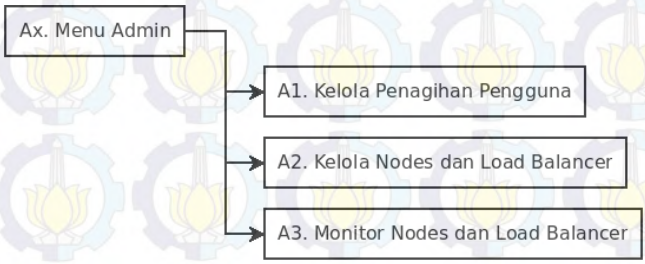
No	Rute	Metode	Hak Akses	Aksi
10	/billings	GET, POST, PUT, DELETE	Admin dan Penyewa	Melakukan manipulasi data tagihan atau billing dari penyewa
11	/billings /*billing-id*/extract	GET	Admin	Membaca spesifikasi pesanan untuk diproses ke rute lain

Proses autentikasi dilakukan dengan melakukan permintaan token pada rute /auth/request. Tokenisasi diimplementasikan berdasarkan standar JSON Web Token (`http://jwt.io`).

3.3.2.2 Frontend Manager

Bagian Frontend Manager dibuat menggunakan platform yang sama dengan yang digunakan oleh Backend. Hanya saja, akses ke Frontend dapat dilakukan melalui peramban Web biasa. Aplikasi Web dibuat menggunakan pustaka Angular.JS (`https://angularjs.org`). Tabel 3.3 menjelaskan rute HTTP yang ada pada Frontend Manager.

Peta situs untuk halaman panel manager pada admin terdapat pada Gambar 3.4, semetara pada sisi pengguna atau penyewa terdapat pada Gambar 3.5.



Gambar 3.4: Peta Situs Panel Admin pada Frontend Manager

Tabel 3.3: Rute pada Frontend Manager

No	Rute	Metode	Hak Akses	Aksi
1	/login	GET, POST	Tidak ada	Menampilkan halaman login dan memrosesnya
2	/admin	GET	Admin	Menampilkan panel manager untuk admin
3	/user	GET	Penyewa	Menampilkan panel manager untuk penyewa
4	/proxy	POST	Semua	Melakuan pemanggilan HTTP ke REST API

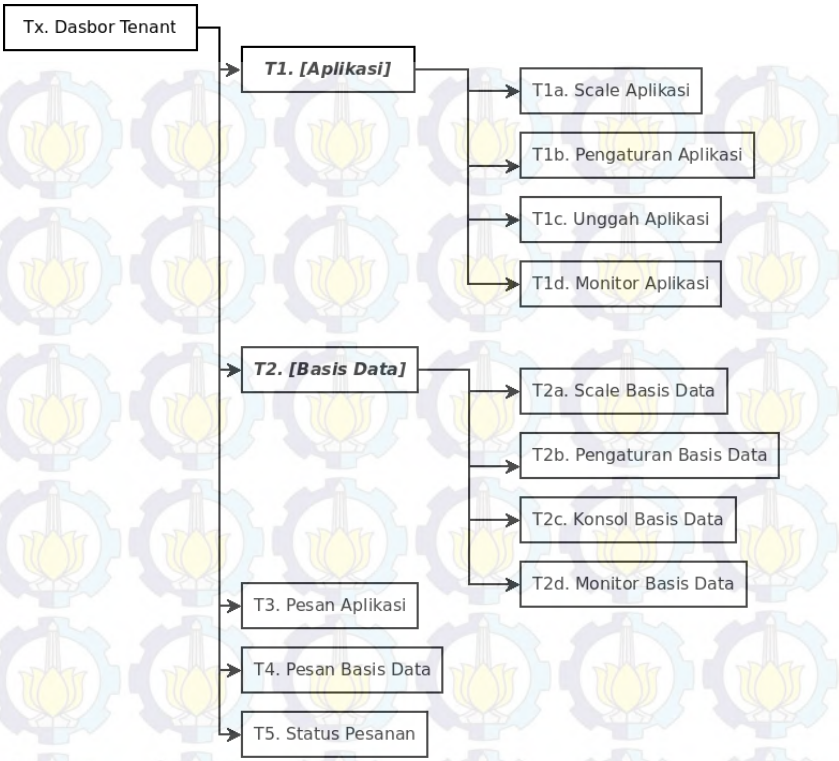
3.3.3 Desain Rinci Load Balancer

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 arsitektur dari Load Balancer tertera pada Gambar 3.6 dan proses diagram contoh proses akses aplikasi dan basis data oleh klien tertera pada Gambar 3.7.

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.

Konfigurasi HAProxy terletak pada berkas /etc/haproxy/haproxy.cfg. Secara umum untuk setiap aplikasi, perlu ada dua bagian pengaturan yang harus diatur:

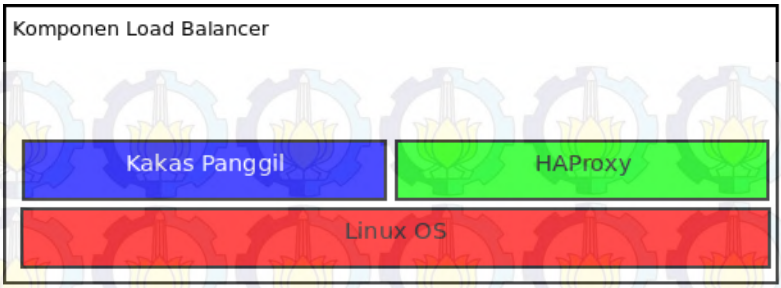
- **HAProxy Frontend** mengatur nama domain dari aplikasi dan



Gambar 3.5: Peta Situs Panel Penyewa pada Frontend Manager

bagaimana HAProxy mengatur koneksi ke domain tersebut. HAProxy menggunakan header HTTP 1.1 bernama "Host" untuk membedakan akses dari satu domain ke domain lainnya yang memiliki aplikasi berbeda.

- **HAProxy Backend** mengatur di Node mana dan port berapa aplikasi akan dikoneksikan setelah permintaan dari Frontend diproses dan bagaimana teknik *load* balancing yang akan dilakukan.



Gambar 3.6: Desain Arsitektur pada Load Balancer

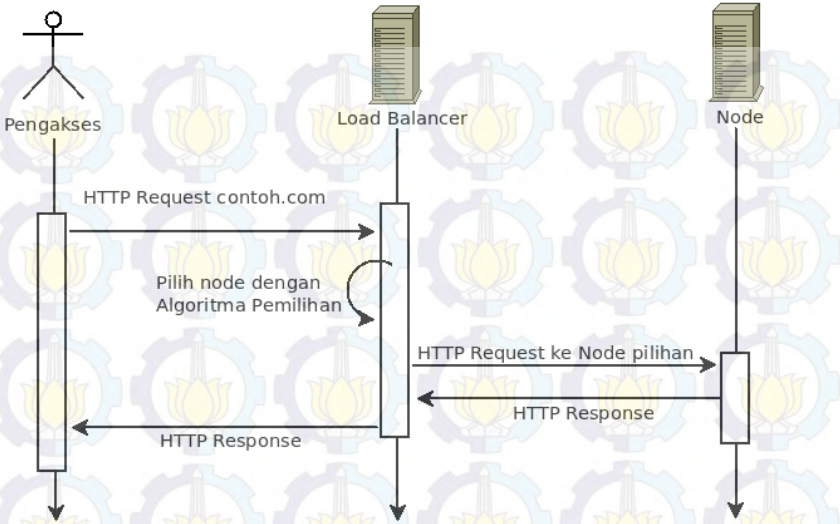
3.3.3.1 Skrip Panggil

Untuk membantu Manager melakukan manipulasi data Load Balancer sesuai dengan kebutuhan, diperlukan beberapa skrip panggil yang ditempatkan di Load Balancer sebagai berikut:

- **register-Frontend** skrip untuk melakukan pendaftaran Frontend baru pada suatu aplikasi.
- **register-Backend** skrip untuk melakukan pendaftaran Backend baru pada suatu aplikasi.
- **unregister-Frontend** skrip untuk melakukan penghapusan data Frontend yang sudah ada.
- **unregister-Backend** skrip untuk melakukan penghapusan data Backend yang sudah ada.
- **reload-proxy** skrip untuk melakukan restart pada HAProxy untuk menerapkan pengaturan yang sudah ada.

3.3.4 Desain Rinci Node

Node merupakan tempat dimana aplikasi Web dan basis data sebenarnya ditempatkan. Node tidak terekspos oleh jaringan luar dan hanya bisa diakses melalui 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 sinkroni-

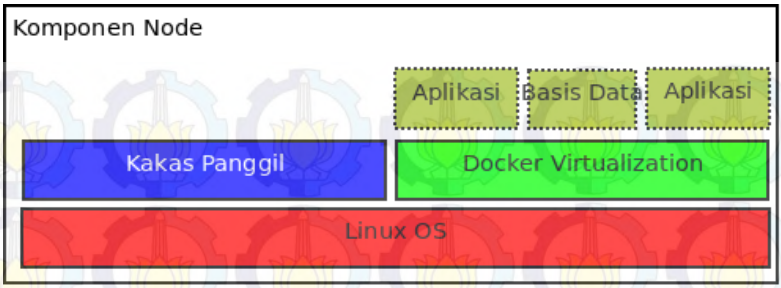


Gambar 3.7: Diagram Interaksi Proses Akses contoh.com yang Dijalankan di Sistem oleh Pengakses.

sasi/replikasi master-master masih belum bisa diaplikasikan secara transparan [17]. Semua data mengenai Node, dan aplikasi Web / basis data yang ter-*hosting* pada suatu Node disimpan pada basis data Hosting Manager. Diagram arsitektur untuk Node tertera pada Gambar 3.8.

Untuk menyimpan beberapa aplikasi dan basis data Web pada satu Node yang sama, setiap Node dibuatkan akun pengguna berbasis PAM (Pluggable Authentication Modules) untuk setiap aplikasi atau basis data. Akun pengguna ini kemudian diatur dengan kuota ruang diska tertentu sesuai dengan yang diinginkan oleh penyewa.

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



Gambar 3.8: Desain Arsitektur setiap Node

mengikuti data aplikasi dari Root Node. Penyewa dapat mengunggah data aplikasi (berupa kode tereksekusi) melalui Git ke Root Node dari aplikasi bersangkutan. Kemudian ketika aplikasi dijalankan, semua Slave Node akan melakukan sinkronisasi data aplikasi dari Root Node.

3.3.4.1 Citra Docker pada Node

Aplikasi atau basis data kemudian 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.

3.3.4.2 Skrip Panggil

Untuk mengelola aplikasi dan basis data yang disimpan pada setiap Node, maka dibuat beberapa skrip panggil berbasis CLI yang bisa dipanggil oleh Manager ketika melakukan aksi dari operasi yang diterima oleh admin atau penyewa. Berikut adalah daftar skrip panggil yang dibuat:

- **create-app** skrip untuk membuat aplikasi baru pada Root Node. Skrip ini tidak perlu dipanggil pada Slave Node.
- **start-app** skrip untuk menjalankan suatu aplikasi pada suatu Node melalui *container* Docker. Skrip dipanggil untuk setiap Node yang memiliki aplikasi bersangkutan.
- **stop-app** skrip untuk menghentikan suatu aplikasi pada suatu Node.
- **status-app** skrip untuk melihat status berjalannya aplikasi pada suatu Node
- **log-app** skrip untuk melihat data berkas log pada suatu aplikasi melalui *container* Docker
- **create-db** skrip untuk membuat basis data baru.
- **start-db** skrip untuk menjalankan suatu basis data pada suatu Node melalui *container* Docker. Skrip dipanggil untuk setiap Node yang memiliki basis data bersangkutan.
- **stop-db** skrip untuk menghentikan suatu basis data pada suatu Node.
- **status-db** skrip untuk melihat status berjalannya basis data pada suatu Node
- **log-db** skrip untuk melihat data berkas log pada suatu basis

data melalui *container* Docker

- **status-*node*** skrip untuk melihat status suatu Node secara umum.
- **log-*node*** skrip untuk membaca berkas log pada Node.

Halaman ini sengaja dikosongkan

BAB 4

IMPLEMENTASI

Bab ini membahas implementasi perancangan sistem PaaS secara rinci. Pembahasan dilakukan untuk setiap komponen yang sudah dijelaskan pada bab sebelumnya yaitu: Manager, Load Balancer dan Node.

4.1 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.
- Peramban *web* Mozilla Firefox.

4.2 Rincian Implementasi Manager

Seperti yang sudah dijelaskan pada subbab 3.3.2, komponen Manager terdiri dari dua bagian: Backend dan Frontend. Berikut adalah penjelasan rincian implementasi dari masing-masing bagian.

4.2.1 Backend Manager

Implementasi Backend manager dibagi berdasarkan rute REST API yang sudah dijelaskan pada Tabel 3.2. Setiap rute dasar dibuat dalam pengendali / controller terpisah.

4.2.1.1 Pengendali `/auth`

Untuk mengatur hak akses pada setiap rute, sistem mengimplementasikan penggunaan token yang wajib dimasukkan oleh pengguna ketika mengakses rute tertentu yang membutuhkan hak akses khusus. Proses untuk mendapatkan token pertama kali dan memerifikasi token dilakukan melalui pengendali `/auth`. Pengendali ini terdiri dari dua sub-rute: rute `/auth/request` untuk proses meminta token, rute `/auth/verify` untuk verifikasi keabsahan token. Implementasi dari kedua sub-rute dapat dilihat pada Tabel 4.1. Proses pembuatan dan verifikasi token menggunakan pustaka jsonwebtoken (`github.com/auth0/node-jsonwebtoken`) pada Node.js.

Tabel 4.1: Implementasi Pengendali `/auth`

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
1	POST /request	-	`Username`, `Password`	`Token`, `Role`	1. Jika `Username` dan `Password` terdaftar • Berikan `token` dan `role`

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
2	POST `/verify`	-	`Token`, `Role`	Boolean (1)	1. Jika `Token` valid dengan `Role` yang benar • Kembalikan TRUE

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

4.2.1.2 Pengendali `/users`

Pengendali ini berfungsi untuk melakukan manipulasi data pengguna penyewa. Rute pengendali `/users` dapat diakses melalui metode GET, POST dan PUT sesuai yang dijelaskan pada Tabel 4.2.

Tabel 4.2: Implementasi Pengendali `/users`

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
1	GET `/`	A	-	Data penyewa (n)	1. Kembalikan data semua penyewa

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
2	GET */userId*	A	-	Data penyewa (1)	1. Kembalikan data penyewa *userId*
3	POST /	A	Data penyewa (1)	Data penyewa (1)	1. Lakukan pembuatan penyewa baru 2. Kembalikan data penyewa yang sudah diberi *userId*
4	PUT */userId*	A	Data penyewa (1)	Data penyewa (1)	1. Ubah data penyewa sesuai yang diminta 2. Kembalikan data penyewa hasil pengubahan

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

4.2.1.3 Pengendali `/apps`

Pengendali ini berfungsi untuk melakukan manipulasi data aplikasi dari setiap penyewa. Tabel 4.3 menjelaskan implementasi sub-rute dari pengendali rute `/apps`.

Pengendali ini memiliki rute khusus bernama POST `/apps/`*`app-id`*`/operasi` yang memiliki berbagai fungsi untuk melakukan operasi pada aplikasi. Daftar operasi dapat dilihat pada Tabel 4.4.

Tabel 4.3: Implementasi Pengendali `/apps`

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
1	GET /	T-A	-	Data aplikasi (n)	1. Apakah pengguna T atau A? • Jika T maka kembalikan data aplikasi milik T • Jika A maka kembalikan semua data aplikasi
2	GET `/appId`	T-A	-	Data aplikasi (1)	1. Apakah pengguna T atau A? • Jika T maka kembalikan data aplikasi `appId` jika memang milik T tersebut. • Jika A maka kembalikan data aplikasi `appId`

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
3	POST /	A	Data aplikasi (1)	Data aplikasi (1)	1. Cari Node yang masih bisa diisi untuk aplikasi baru sesuai jumlah Node yang diminta 2. Jika tersedia cukup jumlah Node • Lakukan pembuatan aplikasi baru • Panggil skrip `create-app` pada Root Node. • Kembalikan data aplikasi yang sudah diberi *`appId`*
4	PUT */`appId`*	A	Data aplikasi (1)	Data aplikasi (1)	1. Ubah data aplikasi sesuai yang diminta 2. Jika perlu melakukan pengubahan pada Root Node • Panggil skrip `modify-app` pada Root Node. 3. Kembalikan data aplikasi hasil pengubahan

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
5	POST `/appId /operation`	T	Operasi	-	*Penjelasan operasi pada aplikasi dapat dilihat pada Tabel 4.4*
6	DELETE `/appId`	A	-	-	1. Hapus data aplikasi 2. Panggil skrip `remove-app` pada Root Node dan Slave Node.

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

Tabel 4.4: Daftar Operasi pada Aplikasi

No	Operasi	Masukan	Langkah Proses
1	start	Menjalankan aplikasi pada semua Node	1. Untuk setiap Node yang menyimpan aplikasi tersebut • Panggil skrip `start-app`.

No	Operasi	Masukan	Langkah Proses
2	stop	Menghentikan aplikasi pada semua Node	1. Untuk setiap Node yang menyimpan aplikasi tersebut • Panggil skrip `stop-app`.
3	statusAllNode	Mengetahui status jalannya aplikasi pada semua Node	1. Untuk setiap Node yang menyimpan aplikasi tersebut • Panggil skrip `status-app`. • Kembalikan hasil pemanggilannya.
4	statusRootNode	Mengetahui status jalannya aplikasi pada Root Node beserta penggunaan penyimpanan dan status repository Git didalamnya.	1. Panggil skrip `status-app` pada Root Node. 2. Kembalikan hasil pemanggilannya.
5	readLog	Membaca berkas log secara raw dari aplikasi.	1. Panggil skrip `log-app` untuk setiap Node. 2. Kembalikan hasil pemanggilannya.

No	Operasi	Masukan	Langkah Proses
6	accessReport	Mengambil data access.log pada aplikasi untuk setiap Node.	1. Untuk setiap Node yang menyimpan aplikasi tersebut • Panggil skrip `log-app` untuk berkas access.log. • Kombinasikan hasilnya jadi satu. 2. Analisa dan kembalikan hasilnya dalam bentuk laporan HTML.
7	reloadLoad-Balancer	Memuat ulang data aplikasi pada *load balancer* agar bisa diakses.	1. Panggil skrip `unregister-Backend` pada *load balancer*. 2. Panggil skrip `unregister-Frontend` pada *load balancer*. 3. Panggil skrip `register-Backend` pada *load balancer*. 4. Panggil skrip `register-Frontend` pada *load balancer*.

4.2.1.4 Pengendali `/dbs`

Pengendali ini berfungsi untuk melakukan manipulasi data basis data dari setiap penyewa. Tabel 4.5 menunjukan teknik implementasinya.

Sama seperti `/apps`, pengendali ini juga memiliki rute khusus operasi yang daftarnya dapat dilihat pada Tabel 4.6.

Tabel 4.5: Implementasi Pengendali `/dbs`

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
1	GET /	T-A	-	Data basis data (n)	1. Apakah pengguna T atau A? • Jika T maka kembalikan data basis data milik T • Jika A maka kembalikan semua data basis data
2	GET `/dbId`	T-A	-	Data basis data (1)	1. Apakah pengguna T atau A? • Jika T maka kembalikan data basis data `dbId` jika memang milik T tersebut. • Jika A maka kembalikan data basis data `dbId`

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
3	POST /	A	Data basis data (1)	Data basis data (1)	1. Lakukan pembuatan basis data baru 2. Panggil skrip `create-db` pada Root Node. 3. Kembalikan data basis data yang sudah diberi *`dbId`*
4	PUT */dbId*	A	Data basis data (1)	Data basis data (1)	1. Ubah data basis data sesuai yang diminta 2. Jika perlu melakukan pengubahan pada Root Node • Panggil skrip `modify-db` pada Root Node. 3. Kembalikan data basis data hasil pengubahan
5	POST */dbId* `/operation`	T	Operasi	-	*Penjelasan operasi pada basis data dapat dilihat pada Tabel 4.6*

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
6	DELETE */dbId*	A	-	-	1. Hapus data basis data 2. Panggil skrip `remove-db` pada Root Node.

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

Tabel 4.6: Daftar Operasi pada Basis Data

No	Operasi	Masukan	Langkah Proses
1	start	Menjalankan basis data pada Node	1. Panggil skrip `start-db` pada Node
2	stop	Menghentikan basis data	1. Panggil skrip `stop-db` pada Node
3	status	Mengetahui status jalannya basis data	1. Panggil skrip `status-db`. 2. Kembalikan hasil pemanggilannya.
4	readLog	Membaca berkas log secara raw dari basis data.	1. Panggil skrip `log-db` 2. Kembalikan hasil pemanggilannya.

4.2.1.5 Pengendali /nodes

Pengendali ini digunakan untuk memanipulasi data Node dan *load balancer* yang terdaftar pada sistem. Penjelasan implementasi pengendali terdapat pada tabel 4.7.

Tabel 4.7: Implementasi Pengendali `/nodes`

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
1	GET `/`	A	-	Data Node (n)	1. Kembalikan data semua Node
2	GET `/nodeId`	A	-	Data Nodes (1)	1. Kembalikan data Node `nodeId`
3	POST `/`	A	Data Node (1)	Data Node (1)	1. Lakukan pembuatan Node baru 2. Kembalikan data Node yang sudah diberi `nodeId`
4	PUT `/nodeId`	A	Data Node (1)	Data nodes (1)	1. Ubah data Node sesuai yang diminta 2. Kembalikan data Node hasil pengubahan

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
5	GET /*nodeId*/status	A	-	Data status Node (1)	1. Panggil kakas **hoster-status-*node*** pada Node bersangkutan. 2. Kembalikan hasilnya ke klien
6	GET /*nodeId* /log /*logFile*	A	-	Berkas log	1. Panggil kakas **hoster-log-*node*** pada Node bersangkutan. 2. Kembalikan hasilnya ke klien

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

4.2.1.6 Pengendali /billings

Pengendali ini digunakan untuk memanipulasi data billing atau tagihan pemesanan yang dikirimkan oleh pengguna dan yang terdaftar pada sistem. Penjelasan implementasi pengendali terdapat pada Tabel 4.8.

Tabel 4.8: Implementasi Pengendali `/billing`

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
1	GET /	A	-	Data billing (n)	1. Kembalikan data semua billing

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

No	Rute	Hak Akses	Masukan	Luaran	Langkah Proses
2	GET `/node-Id`	A	-	Data billing (1)	1. Kembalikan data billing `nodeId`
3	GET `/node-Id /extra-ct`	A	-	Daftar Operasi (n)	1. Ambil data billing `nodeId` 2. Kembalikan daftar operasi untuk menerapkan billing tersebut.
4	POST `/`	T	Data billing (1)	Data billing (1)	1. Lakukan pembuatan billing baru 2. Kembalikan data billing yang sudah diberi `nodeId`
5	PUT `/node-Id`	A	Data billing (1)	Data billing (1)	1. Ubah data billing sesuai yang diminta 2. Kembalikan data billing hasil pengubahan

Keterangan: *T: Hak Akses Penyewa, A: Hak Akses Admin, (1) Data Tunggal, (n) Data Jamak*

4.2.2 Frontend Manager

Frontend Manager dibuat dalam entitas *server* Web terpisah dengan Backend. Bagian Frontend hanya menghubungkan antarmuka pengguna (baik penyewa maupun administrator) dengan REST API pada Backend. Terdapat dua rute akses http pada Frontend, yaitu: halaman antarmuka aplikasi pada rute `/admin` (Antarmuka Administrator) dan `/user` (Antarmuka Penyewa) diatur melalui pengendali sisi klien menggunakan pustaka AngularJS. Sub-bab ini membahas mengenai implementasi peta situs dan

pengendali pada antarmuka aplikasi.

4.2.2.1 Antarmuka Administrator

Tabel 4.9 menjelaskan mengenai implementasi peta situs (dari gambar 3.4) dari halaman antarmuka administrator pada Frontend dengan rute `/admin` serta keterkaitannya dengan halaman pada peta situs dan kasus penggunaan (*use case*) yang diimplementasikan. Setiap implementasi memiliki sub-rute yang diimplementasikan menggunakan AngularJS. Akses ke halaman spesifik dapat dilakukan melalui akses ke `/admin/#sub-rute` melalui peramban Web klien.

Tabel 4.9: Implementasi Peta Situs pada Antarmuka Panel Administrator

No	Halaman	Sub-rute	Use Case	Nama Pengendali	Operasi
1	Ax	/	-	dashboard	-
2	A1	`/billing`	UC09	billing	**getBilling**: Ambil data billing. **setujuiBilling**: Tandai billing sebagai disetujui. **hapusBilling**: Batalkan dan hapus billing. **prosesBilling**: Lakukan operasi untuk menerapkan billing (melalui rute Backend `/billings/billingId/extract`).

No	Halaman	Sub-rute	Use Case	Nama Pengen-dali	Operasi
3	A2	/	UC08	nodes	**getNodes**: Ambil data Node dan *load balancer*. **addNode**: Membuat Node atau *load balancer* baru. **editNode**: Menyunting Node atau *load balancer* yang sudah ada.
4	A3	/	UC10	monitor	**getNodes**: Ambil data Node dan *load balancer*. **getNodesStatus**: Ambil data rincian status pada Node atau *load balancer* tersebut. **downloadLog**: Ambil berkas log dari suatu Node atau *load balancer*.

4.2.2.2 Antarmuka Penyewa

Tabel 4.10 menjelaskan mengenai implementasi halaman antarmuka panel penyewa pada Frontend.

Tabel 4.10: Implementasi Peta Situs pada Antarmuka Panel Penyewa

No	Halaman	Sub-rute	Use Case	Nama Pengendali	Operasi
1	Tx, T1, T2	`/`	-	dashboard	-
2	T1a	`/apps/`*`appId`*`/scale`	UC09	scaleApps	**calculatePrice**: menghitung harga *scaling*. **sendBilling**: mengirimkan data billing melalui rute Backend POST `/billings`.
3	T1c	`/apps/`*`appId`*`/upload`	UC06	uploadApps	**getAppStatus**: mengambil data keadaan berjalannya aplikasi. Digunakan untuk mengecek jumlah Node yang menjalankan aplikasi.
4	T3	`/billing/order/app`	UC01	billingOrderApp	**calculatePrice**: menghitung harga pemesanan aplikasi. **sendBilling**: mengirimkan data billing melalui rute Backend POST `/billings`.

No	Halaman	Sub-rute	Use Case	Nama Pengendali	Operasi
5	T1b	`/apps/appId /configure`	UC04	configureApps	**getAppData**: mengambil data aplikasi yang diatur. **startApp**: menjalankan aplikasi. Panggil operasi `start` melalui rute Backend. **stopApp**: menghentikan aplikasi. Panggil operasi `stop` melalui rute Backend. **getAppStatus**: mengambil data keadaan berjalannya aplikasi. Digunakan untuk mengecek jumlah Node yang menjalankan aplikasi. **reloadLoadBalancer**: memuat ulang *load balancer* untuk aplikasi yang diatur. Panggil operasi `reloadLoadBalancer` melalui rute Backend. **applyConfiguration**: menerapkan pengaturan pada aplikasi.

No	Halaman	Sub-rute	Use Case	Nama Pengendali	Operasi
6	T1d	`/apps/`*`appId`*`/monitor`	UC05	monitorApps	**getAppData**: mengambil data aplikasi yang diatur. **getNodeStatus**: mengambil data keadaan berjalannya aplikasi. Digunakan untuk mengecek jumlah Node yang menjalankan aplikasi. **downloadLog**: membaca data log pada aplikasi. **webServerReport**: membaca rangkuman akses (berkas log access.log) pada aplikasi.
7	T2a	`/dbs/`*`dbId`*`/scale`	UC09	scaleDbs	**calculatePrice**: menghitung harga *scaling*. **sendBilling**: mengirimkan data billing melalui rute Backend POST `/billings`.
8	T2c	`/dbs/`*`dbId`*`/console`	UC06	uploadDbs	**sendCommand**: mengirimkan perintah pada konsol basis data yang sedang berjalan.

No	Halaman	Sub-rute	Use Case	Nama Pengendali	Operasi
9	T2b	`/dbs/dbId /configure`	UC04	configureDbs	**getDbData**: mengambil data basis data yang diatur. **startDb**: menjalankan basis data. Panggil operasi `start` melalui rute Backend. **stopDb**: menghentikan basis data. Panggil operasi `stop` melalui rute Backend. **getDbStatus**: mengambil data keadaan berjalannya basis data. Digunakan untuk mengecek jumlah Node yang menjalankan basis data. **applyConfiguration**: menerapkan pengaturan pada basis data.
10	T2d	`/dbs/dbId /monitor`	UC05	monitorDbs	**getDbData**: mengambil data basis data yang diatur. **getNodeStatus**: mengambil data keadaan berjalannya basis data. Digunakan untuk mengecek jumlah Node yang menjalankan basis data. **downloadLog**: membaca data log pada basis data.

No	Halaman	Sub-rute	Use Case	Nama Pengendali	Operasi
11	T4	`/billing/order/db`	UC01	billing-OrderDb	**calculatePrice**: menghitung harga pemesanan basis data. **sendBilling**: mengirimkan data billing melalui rute Backend POST `/billings`.
12	T5	`/billing/history`	UC01	billing-History	**getBillings**: mengambil data billing yang dimiliki oleh pengguna.

4.3 Rincian Implementasi Load Balancer

Subbab ini menjelaskan rincian implementasi sistem pada sisi *load balancer*.

4.3.1 Kakas Panggil

Tabel 4.11 menjelaskan implementasi kakas panggil pada *load balancer*.

Tabel 4.11: Implementasi Kakas Panggil pada Load Balancer

No	Kakas	Argumen	Proses
1	**register-*frontend***	`appId, domainName`	Daftarkan `appId` sebagai *frontend* pada berkas konfigurasi HAProxy dengan nama domain `domainName`.
2	**register-*backend***	`appId, ipNode, port`	Daftarkan alamat `ipNode:port` sebagai *peer* dari aplikasi `appId` pada konfigurasi HAProxy.
3	**unregister-*frontend***	`appId`	Menghapus definisi *frontend* untuk aplikasi `appId` pada konfigurasi HAProxy
4	**unregister-*backend***	`appId`	Menghapus definisi *backend* untuk aplikasi `appId` pada konfigurasi HAProxy
5	**reload-proxy**	-	Memuat ulang HAProxy dan menerapkan konfigurasi terbaru

4.4 Rincian Implementasi Node

Subbab ini menjelaskan rincian implementasi sistem pada sisi Node.

4.4.1 Citra Cakram Docker

Tabel 4.12 menjelaskan rincian citra cakram Docker yang dibangun beserta aplikasi yang ada di dalamnya. Setiap citra cakram memi-

liki aturan khusus dalam proses penempatan berkas aplikasi untuk mempermudah penyewa menempatkan berkas aplikasi agar sesuai dan bisa dijalankan.

Tabel 4.12: Rincian Implementasi Citra Cakram Docker pada Node

No	Nama	Layanan	Penempatan Aplikasi
1	nodejs-nodemon	1. Nginx 2. Node.js dan npm 3. Nodemon	Aplikasi utama harus berada pada berkas `app.js` dan berjalan pada port 3000. Paket yang dibutuhkan didefinisikan pada berkas `package.json` untuk kemudian dipasangkan pada saat proses `start` aplikasi.
2	php55-fpm	1. Nginx 2. PHP 5.5 3. Modul php5-fpm 4. Composer	Aplikasi utama harus berada pada berkas `index.php`. Paket yang dibutuhkan didefinisikan pada berkas `composer.json` untuk kemudian dipasangkan pada saat proses `start` aplikasi menggunakan `composer`

No	Nama	Layanan	Penempatan Aplikasi
3	python27-gunicorn	1. Nginx 2. Python 2.7 3. virtualenv 4. Gunicorn	Nama pustaka utama WSGI didefinisikan pada berkas `gunicorn-args.txt`. Paket yang dibutuhkan didefinisikan pada berkas `requirements.txt` menggunakan format `pip`. Paket akan dipasang melalui `virtualenv` pada saat aplikasi pertama kali dijalankan.
4	ruby19-thin	1. Nginx 2. Ruby 1.9 3. Thin 4. Bundle	Paket yang dibutuhkan didefinisikan pada berkas `Gemfile` untuk dipasang menggunakan perintah `gem install` pada saat menjalankan aplikasi pertama kalinya. Berkas `config.ru` akan menjadi titik awal dari aplikasi dan harus memuat modul Bundler agar dapat memuat paket yang diletakkan pada direktori `vendor/bundle`.
5	mysql-single	1. MySQL 5.5	*Bukan Citra untuk Aplikasi*

Hampir semua komponen yang ada di dalam setiap citra cakram menggunakan pengaturan bawaan dari upstream (pengaturan yang disediakan oleh Ubuntu), kecuali pada pengaturan *server* Web nginx yang diatur agar hanya memiliki satu *worker* (bawaannya ada

empat) dan MySQL yang menggunakan MyISAM agar menghemat memori.

4.4.2 Kakas Panggil Node

Tabel 4.13 menjelaskan implementasi kakas panggil pada Node.

Tabel 4.13: Implementasi Kakas Panggil pada Node

No	Kakas	Argumen	Proses
1	**start-app**	`appId`, `appType`, `memoryLimit`, `rootNode`	1. Apakah `rootNode` didefinisikan? • Jika iya, ambil data aplikasi melalui `git clone` atau `git pull` dari *root node*. 2. Jalankan aplikasi dari data aplikasi melalui perintah `docker run` dengan jenis citra sesuai `appType`
2	**stop-app**	`appId`	1. Hentikan aplikasi dari data aplikasi melalui perintah `docker rm`.
3	**create-app**	`appId`, `quota`	1. Buat direktori home untuk penempatan aplikasi. 2. Atur quota pada direktori tersebut dengan perintah `setquota`. 3. Buat direktori Git pertama pada aplikasi menggunakan perintah `git --bare init`.
4	**remove-app**	`appId`	1. Hapus direktori home dari aplikasi beserta datanya.

No	Kakas	Argumen	Proses
5	**status-app**	`appId, isRootNode`	1. Ambil data status aplikasi melalui perintah `docker inspect`. 2. Jika `isRootNode` di-set • Ambil data penggunaan kuota diska menggunakan perintah `du`. • Ambil data status log Git melalui perintah `git log`.
6	**log-app**	`appId, logFile`	1. Ambil berkas `logFile` melalui direktori log aplikasi yang sedang berjalan.
7	**start-db**	`dbId, dbType, memoryLimit`	1. Jalankan basis data dari data basis data melalui perintah `docker run` dengan jenis citra sesuai `dbType`
8	**stop-db**	`dbId`	1. Hentikan basis data dari data basis data melalui perintah `docker rm`.
9	**create-db**	`dbId, quota`	1. Buat direktori home untuk penempatan basis data. 2. Atur quota pada direktori tersebut dengan perintah `setquota`.
10	**remove-db**	`dbId`	1. Hapus direktori home dari basis data beserta datanya.

No	Kakas	Argumen	Proses
11	**status-db**	`dbId`, `isRootNode`	1. Ambil data status basis data melalui perintah `docker inspect`. 2. Jika `isRootNode` di-set • Ambil data penggunaan kuota diska menggunakan perintah `du`.
12	**log-db**	`dbId`, `logFile`	1. Ambil berkas `logFile` melalui direktori log basis data yang sedang berjalan.

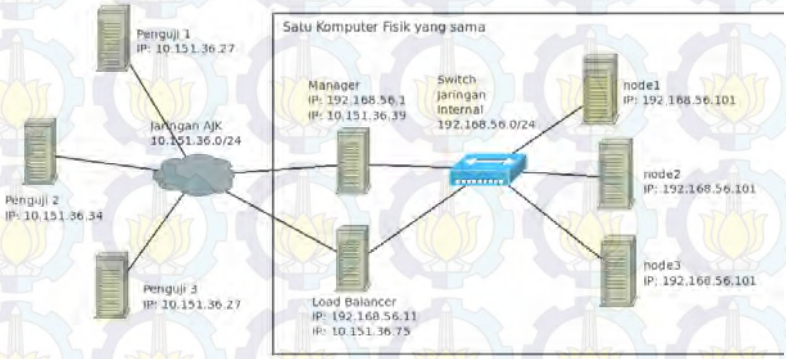
BAB 5

PENGUJIAN DAN EVALUASI

5.1 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. Sementara itu, tiga komputer penguji merupakan komputer fisik di luar komputer yang dilakukan dalam proses implementasi. 5.1 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.



Gambar 5.1: Toplogi Jaringan dalam Pengujian

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 256 MB
        * Hard Disk 8 GB
    - Perangkat Lunak
        * Ubuntu Linux 14.04 LTS
        * Node.js 0.10
        * 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

Untuk menyambungkan seluruh komputer, dilakukan pengaturan alamat IP sebagai berikut:

- Manager pada 192.168.56.1 (dan 10.151.36.39 jika diakses

dari luar).
- Load Balancer pada 192.168.56.11 (dan 10.151.36.75 jika diaksed dari luar).
- Node 1 pada 192.168.56.101.
- Node 2 pada 192.168.56.102.
- Node 3 pada 192.168.56.103.
- Komputer Penguji 1 pada 10.151.36.27
- Komputer Penguji 2 pada 10.151.36.34
- Komputer Penguji 3 pada 10.151.36.40

5.2 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.

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.

5.2.1 Uji Unit Fungsionalitas

Uji unit dilakukan dengan melakukan uji coba request HTTP ke REST API pada Backend Manager untuk beberapa pengendali dasar pada Backend dan meninjau apakah nilai yang dikembalikan se-

suai dengan yang diharapkan . Tabel 5.1 menunjukkan rancangan setiap aksi uji unit dan hasil atau nilai yang diharapkan. Implementasi uji unit dibuat menggunakan pustaka **jasmine-*node*** (`https://github.com/mhevery/jasmine-node`) berbasis Javascript.

Tabel 5.1: Implementasi Uji Unit

No	Pengendali	Uji Coba	Hasil Harapan
1	`/auth`	Mengambil token untuk akun administrator	Token didapatkan
		Mengambil data `/users` melalui token untuk administrator	Tidak ada penolakan token
		Mengambil data `/users` melalui token yang salah	Ada penolakan token
		Mengambil token untuk akun pengguna biasa / penyewa	Token didapatkan
		Mengambil data `/users` melalui token untuk pengguna	Ada penolakan token
2	`/users`	Mendaftarkan pengguna ke sistem	Mendapatkan identitas pengguna
		Mendapatkan token dari pengguna baru	Token didapatkan
		Dapat melihat data pengguna melalui rute `/users/userId`	Data didapatkan
		Dapat mengubah kata sandi	-
		Dapat mendapatkan token dari pengguna dengan kata sandi baru	Token didapatkan

No	Pengendali	Uji Coba	Hasil Harapan
3	`/apps`	Melakukan login sebagai pengguna	Mendapatkan token pengguna
		Melakukan pemesanan aplikasi baru ke billing (dengan 1 Node, RAM 64 MB, ruang cakram 50 MB)	Mendapatkan identitas billing
		Melakukan login sebagai admin	Mendapatkan token admin
		Menyetujui pemesanan aplikasi baru	-
		Mendapatkan `extract` dari billing	Mendapatkan data `extract`
		Melakukan operasi `extract` untuk membuat aplikasi baru	Mendapatkan identitas aplikasi dan alamat clone GIT
		Mendapatkan status aplikasi sebelum dijalankan	Mendapatkan status aplikasi belum berjalan
		Melakukan `git clone` pada alamat `gitAddress`	Tidak ada galat
		Melakukan pengisian data aplikasi, lalu `git commit` dan `git push` ke *root*-Node	Tidak ada galat
		Menjalankan aplikasi	Tidak ada galat
		Mendapatkan status aplikasi setelah dijalankan	Mendapatkan status aplikasi berjalan pada satu Node

No	Pengendali	Uji Coba	Hasil Harapan
		Melakukan operasi `reloadLoadBalancer` pada aplikasi	Tidak ada galat
		Menghentikan aplikasi	Tidak ada galat
		Melakukan pemesanan meningkatkan aplikasi ke billing (ditingkatkan ke tiga Node, RAM 128 MB, ruang cakram 200 MB)	Mendapatkan `billingId`
		Menyetujui pemesanan peningkatan aplikasi	-
		Mendapatkan `extract` dari billing	Mendapatkan `extract`
		Melakukan operasi `extract` untuk meningkatkan aplikasi	Komponen yang ditingkatkan sudah diterapkan
		Menjalankan aplikasi dengan tiga Node	Tidak ada galat
		Mendapatkan status aplikasi setelah dijalankan	Mendapatkan status aplikasi berjalan pada tiga Node
4	`/dbs`	Melakukan login sebagai pengguna	Mendapatkan token pengguna
		Melakukan pemesanan basis data baru ke billing (dengan RAM 64 MB, ruang cakram 100 MB)	Mendapatkan identitas billing
		Melakukan login sebagai admin	Mendapatkan token admin
		Menyetujui pemesanan basis data baru	-

No	Pengendali	Uji Coba	Hasil Harapan
		Mendapatkan `extract` dari billing	Mendapatkan `extract`
		Melakukan operasi `extract` untuk membuat basis data baru	Mendapatkan identitas basis data
		Mendapatkan status basis data sebelum dijalankan	Mendapatkan status basis data belum berjalan
		Menjalankan basis data	Tidak ada galat
		Mengisi data contoh pada basis data	Tidak ada galat
		Mendapatkan status basis data setelah dijalankan	Mendapatkan status basis data berjalan
		Menghentikan basis data	Tidak ada galat
		Melakukan pemesanan meningkatkan basis data ke billing (ditingkatkan ke RAM 128 MB, ruang cakram 200 MB)	Mendapatkan `billingId`
		Menyetujui pemesanan peningkatan basis data	-
		Mendapatkan `extract` dari billing	Mendapatkan `extract`
		Melakukan operasi `extract` untuk meningkatkan basis data	Komponen yang ditingkatkan sudah diterapkan
		Menjalankan basis data	Tidak ada galat
		Mendapatkan status basis data setelah dijalankan	Mendapatkan status basis data berjalan

No	Pengendali	Uji Coba	Hasil Harapan
5	`/nodes`	Melakukan login sebagai administrator	Mendapatkan token admin
		Melakukan penambahan Node	Mendapatkan `nodeId`
		Mengubah data pada Node	Data terubah

5.2.2 Uji Kapasitas

Uji kapasitas dilakukan secara otomatis menggunakan skrip. Berikut adalah langkah-langkah dalam proses pengujian ini untuk setiap percobaannya:

1. Membuat lima akun penyewa secara acak
2. Pada akun penyewa tersebut, dilakukan pembuatan aplikasi atau basis data (atau kombinasi keduanya) baru dengan jumlah Node (khusus aplikasi), jumlah platform (ada empat pilihan),dan jumlah memori (antara 64 MB, 96 MB atau 128 MB) secara acak. Jumlah ruang penyimpanan ditetapkan menjadi 128 MB.
3. Jika pembuatan berhasil, ulang langkah pertama. Jika pembuatan gagal karena ketidak cukupan jumlah Node dan jumlah Node dari aplikasi tersebut berjumlah satu, akhiri pengujian.

Setiap entitas aplikasi dan basis data akan diberi sebuah data contoh yang sudah disiapkan sebelumnya. Setelah pengujian berlangsung, dilakukan proses merangkum daftar aplikasi yang berjalan pada setiap Node untuk kemudian diuji pada uji performa.

5.2.3 Uji Performa

Uji performa dilakukan pada aplikasi yang sudah ditambahkan melalui uji kapasitas pada setiap percobaan. Perangkat lunak yang digunakan dalam pengujian adalah Apache Benchmark (`http://httpd.apache.org/docs/2.2/programs/ab.html`).

Pengujian dilakukan secara bersamaan untuk semua aplikasi dan basis data yang dibuat dalam satu percobaan. Metode *benchmark* dilakukan melalui empat komputer penguji dengan mengirimkan

akses permintaan HTTP sebanyak 500 kali dalam 50 socket secara bersamaan. Hasil pengujian di setiap komputer uji diolah menjadi satu untuk kemudian dilaporkan dalam bentuk tabel.

5.2.4 Aplikasi Contoh

Terdapat dua jenis aplikasi contoh yang dibuat untuk kebutuhan pengujian performa:

- **Dengan Basis Data**, merupakan aplikasi contoh yang terhubung dengan suatu *server* MySQL yang berbeda antara satu sama lain.
- **Tanpa Basis Data**, merupakan aplikasi contoh yang hanya membaca data JSON yang terletak di berkas *server*.

Pembedaan antara kedua jenis tersebut dimaksudkan karena pembatasan kemampuan penyewaan MySQL yang hanya maksimum terdiri dari satu Node sementara perlu adanya pengujian performa murni aplikasi jika hanya dijalankan pada lebih dari satu Node.

Aplikasi contoh hanya melayani satu rute GET dan menampilkan data berupa JSON tanpa menggunakan templat HTML atau CSS.

5.2.5 Basis Data Contoh

Basis data contoh terdiri dari sebuah basis data MySQL yang berisikan sebuah tabel (bernama `test`) yang berisikan tiga kolom sebagai berikut:

- Kolom `id` berisi nomor yang diisi otomatis oleh MySQL.
- Kolom `val` berisi sebuah kalimat acak yang diambil dari Lorem Ipsum.
- Kolom `description` berisi sebuah paragraf acak yang diambil dari Lorem Ipsum.

5.3 Hasil Uji Coba dan Evaluasi

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

5.3.1 Uji Unit Fungsionalitas

Hasil uji unit fungsionalitas dijelaskan pada tabel 5.2 berikut. Contoh tangkapan layar dari hasil uji unit pengendali `/apps` dan `/dbs` dapat dilihat pada 5.2 dan 5.3.

Tabel 5.2: Hasil Eksekusi Uji Unit Fungsionalitas

No	Pengendali	Uji Coba	Hasil
1	`/auth`	Mengambil token untuk akun administrator	Sukses - 17 ms
		Mengambil data `/users` melalui token untuk administrator	Sukses - 13 ms
		Mengambil data `/users` melalui token yang salah	Sukses - 12 ms
		Mengambil token untuk akun pengguna biasa / penyewa	Sukses - 85 ms
		Mengambil data `/users` melalui token untuk pengguna	Sukses - 11 ms
2	`/users`	Mendaftarkan pengguna ke sistem	Sukses - 104 ms
		Mendapatkan token dari pengguna baru	Sukses - 84 ms
		Dapat melihat data pengguna melalui rute `/users/userId`	Sukses - 13 ms
		Dapat mengubah kata sandi	Sukses - 82 ms
		Dapat mendapatkan token dari pengguna dengan kata sandi baru	Sukses - 83 ms

No	Pengendali	Uji Coba	Hasil
3	`/apps`	Melakukan login sebagai pengguna	Sukses - 150 ms
		Melakukan pemesanan aplikasi baru ke billing (dengan 1 *node*, RAM 64 MB, ruang cakram 50 MB)	Sukses - 36 ms
		Melakukan login sebagai admin	Sukses - 12 ms
		Menyetujui pemesanan aplikasi baru	Sukses - 13 ms
		Mendapatkan `extract` dari billing	Sukses - 12 ms
		Melakukan operasi `extract` untuk membuat aplikasi baru	Sukses - 917 ms
		Mendapatkan status aplikasi sebelum dijalankan	Sukses - 526 ms
		Melakukan `git clone` pada alamat `gitAddress`	Sukses - 494 ms
		Melakukan pengisian data aplikasi, lalu `git commit` dan `git push` ke *root node*	Sukses - 1.000 ms
		Menjalankan aplikasi	Sukses - 905 ms
		Mendapatkan status aplikasi setelah dijalankan	Sukses - 945 ms
		Melakukan operasi `reloadLoadBalancer` pada aplikasi	Sukses - 2.361 ms
		Menghentikan aplikasi	Sukses - 1.263 ms

No	Pengendali	Uji Coba	Hasil
		Melakukan pemesanan meningkatkan aplikasi ke billing (ditingkatkan ke 3 *node*, RAM 128 MB, ruang cakram 200 MB)	Sukses - 12 ms
		Menyetujui pemesanan peningkatan aplikasi	Sukses - 11 ms
		Mendapatkan `extract` dari billing	Sukses - 463 ms
		Melakukan operasi `extract` untuk meningkatkan aplikasi	Sukses - 3.546 ms
		Menjalankan aplikasi dengan 3 *node*	Sukses - 1.740 ms
		Mendapatkan status aplikasi setelah dijalankan	Sukses - 4.763 ms
4	`/dbs`	Melakukan login sebagai pengguna	Sukses - 124 ms
		Melakukan pemesanan basis data baru ke billing (dengan RAM 64 MB, ruang cakram 100 MB)	Sukses - 34 ms
		Melakukan login sebagai admin	Sukses - 11 ms
		Menyetujui pemesanan basis data baru	Sukses - 12 ms
		Mendapatkan `extract` dari billing	Sukses - 12 ms
		Melakukan operasi `extract` untuk membuat basis data baru	Sukses - 664 ms

No	Pengendali	Uji Coba	Hasil
		Mendapatkan status basis data sebelum dijalankan	Sukses - 449 ms
		Menjalankan basis data	Sukses - 10.893 ms
		Mengisi data contoh pada basis data	Sukses - 745 ms
		Mendapatkan status basis data setelah dijalankan	Sukses - 1.103 ms
		Menghentikan basis data	Sukses - 630 ms
		Melakukan pemesanan meningkatkan basis data ke billing (ditingkatkan ke RAM 128 MB, ruang cakram 200 MB)	Sukses - 21 ms
		Menyetujui pemesanan peningkatan basis data	Sukses - 13 ms
		Mendapatkan `extract` dari billing	Sukses - 12 ms
		Melakukan operasi `extract` untuk meningkatkan basis data	Sukses - 483 ms
		Menjalankan basis data	Sukses - 10.903 ms
		Mendapatkan status basis data setelah dijalankan	Sukses - 481 ms
5	`/nodes`	Melakukan login sebagai admin	Sukses - 56 ms
		Melakukan penambahan *node*	Sukses - 18 ms
		Mengubah data pada *node*	Sukses - 13 ms

Pada hasil uji coba fungsionalitas tersebut, didapatkan bahwa

```
/apps API testing - 19195 ms
    should able to log in as an user - 150 ms
    should able to order new app inside billing as user - 36 ms
    should able to log in as an admin - 12 ms
    should able to accept new billing as admin - 13 ms
    should able to extract new Billing for next operation - 12 ms
    should able to execute extraction data to create new apps - 917 ms
    should able to get app status before starting - 526 ms
    should able to do 'git clone' for apps - 494 ms
    should able to commit, push sample application from ../../../docker/nodejs-nodemon/sample - 1000 ms
    should able to start the application - 905 ms
    should able to get app status after starting - 945 ms
    should able to register running app inside load balancer - 2361 ms
    should able to stop the application - 1263 ms
    should able to order app scaling inside billing as user - 12 ms
    should able to accept new billing as admin - 11 ms
    should able to extract new Billing for next operation - 19 ms
    should able to execute extraction data to scale-up for apps - 463 ms
    should able to start the application with scaled-up resource - 3546 ms
    should able to get app status after starting with scaled up resource - 1740 ms
    should able to register running app inside load balancer - 4763 ms

Finished in 19.198 seconds
20 tests, 37 assertions, 0 failures, 0 skipped
```

Gambar 5.2: Tangkapan Layar **jasmine-*node*** dalam Melakukan Uji Unit pengendali `/apps`

```
/dbs API testing - 26590 ms
    should able to log in as an user - 124 ms
    should able to order new db inside billing as user - 34 ms
    should able to do request token access for admin - 11 ms
    should able to accept new billing as admin - 12 ms
    should able to extract new Billing for next operation - 12 ms
    should able to execute extraction data to create new dbs - 664 ms
    should able to get db status before starting - 449 ms
    should able to start the database - 10893 ms
    should able to insert some sample data into database - 745 ms
    should able to get db status after starting - 1103 ms
    should able to stop the database - 630 ms
    should able to order db scaling inside billing as user - 21 ms
    should able to accept new billing as admin - 13 ms
    should able to extract new Billing for next operation - 12 ms
    should able to execute extraction data to scale the db - 483 ms
    should able to start the database - 10903 ms
    should able to get db status after starting - 481 ms

Finished in 26.595 seconds
17 tests, 31 assertions, 0 failures, 0 skipped
```

Gambar 5.3: Tangkapan Layar **jasmine-*node*** dalam Melakukan Uji Unit Pengendali `/dbs`

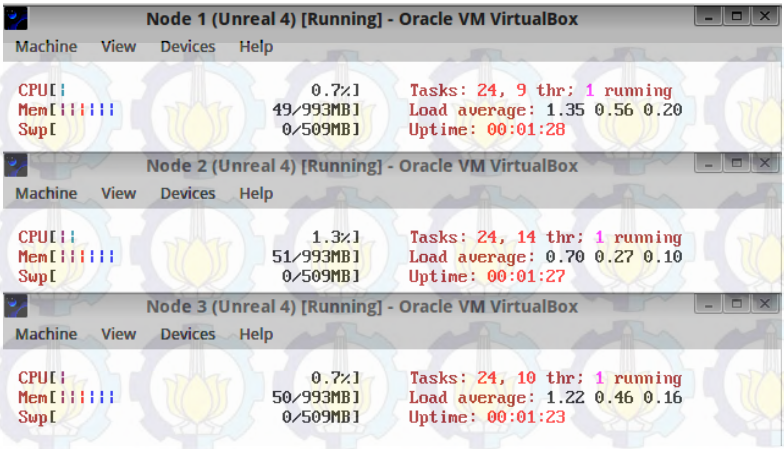
Backend pada Manager sudah mengimplementasikan keseluruhan fungsionalitas dengan baik sesuai dengan yang diharapkan. Sebagian besar aksi yang dilakukan pada Backend dapat dilakukan dengan waktu dibawah 100 ms.

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

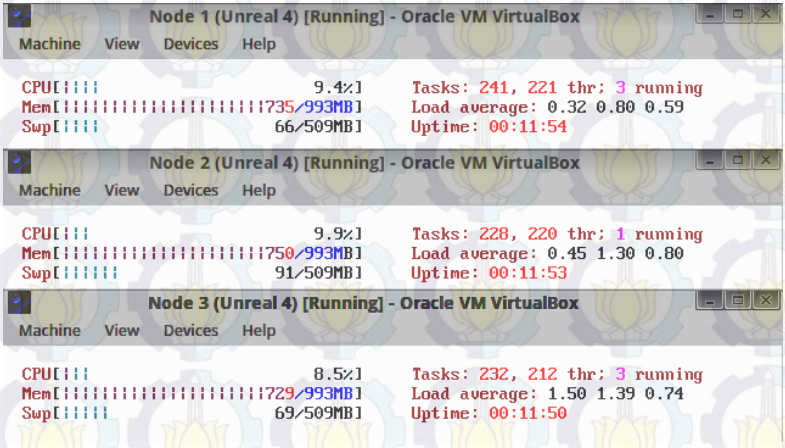
5.3.2 Uji Kapasitas dan Performa

Uji kapasitas pada setiap kali percobaan rata-rata berlangsung sekitar 8 - 15 menit. Sebelum pengujian dilakukan (dalam keadaan Node baru dalam keadaan dihidupkan), masing-masing Node menggunakan memori sebanyak 50 MB dan penggunaan CPU yang cenderung *idle* 0%. Sementara setelah pengujian dilakukan, penggunaan memori RAM rata-rata mencapai 650 - 800 MB dengan penggunaan CPU yang sekitar 10 - 15% pada masing-masing Node. Contoh tampilan aplikasi `htop` sebelum dan sesudah uji pecobaan satu dapat dilihat pada 5.4 dan 5.5. Sementara ketika uji performa berlangsung, Node dan Load Balancer mengalami peningkatan penggunaan CPU mencapai 100% seperti pada Gambar 5.6.

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. Daftar aplikasi dan hasil uji performa pada setiap percobaan dapat dilihat pada tabel 5.3.



Gambar 5.4: Keadaan Node dan Load Balancer SEBELUM Uji Kapasitas Percobaan Pertama Dilakukan (Komputer dalam Keadaan Baru Dihidupkan)



Gambar 5.5: Keadaan Node dan Load Balancer SETELAH Uji Kapasitas Percobaan Pertama Dilakukan

Tabel 5.3: Daftar Aplikasi pada Uji Kapasitas Dengan Hasil Uji Performa

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
Percobaan ke-1									
1	benton	php55-fpm	1	64 MB	Tidak	-	536	30	36%
2	dayton	python27-gunicorn	1	96 MB	Tidak	-	756	31	50%
3	braulio	python27-gunicorn	1	96 MB	Ya	96 MB	616	33	41%
4	junius	php55-fpm	1	128 MB	Tidak	-	580	30	39%
5	denis	nodejs-nodemon	1	128 MB	Tidak	-	573	31	38%
6	romaine	python27-gunicorn	3	64 MB	Ya	64 MB	1499	36	100%
7	brett	python27-gunicorn	2	64 MB	Ya	64 MB	1131	30	75%

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
8	monserrat	nodejs-nodemon	2	96 MB	Ya	64 MB	593	30	40%
9	liliana	nodejs-nodemon	2	128 MB	Tidak	-	1349	31	90%
10	ezequiel	ruby19-thin	2	96 MB	Ya	128 MB	1195	33	80%
11	brice	python27-gunicorn	2	96 MB	Tidak	-	914	47	61%
12	paula	python27-gunicorn	1	128 MB	Tidak	-	792	30	53%
13	stevie	ruby19-thin	1	64 MB	Ya	128 MB	685	30	46%
14	paris	php55-fpm	1	96 MB	Ya	64 MB	493	30	33%
Percobaan ke-2									
15	marjory	ruby19-thin	1	128 MB	Tidak	-	1490	30	99%

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
16	laurence	python27-gunicorn	1	96 MB	Tidak	-	778	30	52%
17	tito	php55-fpm	1	96 MB	Tidak	-	577	30	38%
18	joany	python27-gunicorn	1	64 MB	Tidak	-	990	30	66%
19	norbert	python27-gunicorn	2	96 MB	Ya	128 MB	1500	30	100%
20	buster	ruby19-thin	3	128 MB	Tidak	-	1500	39	100%
21	terrill	python27-gunicorn	3	96 MB	Ya	64 MB	1499	38	100%
22	jordane	ruby19-thin	2	96 MB	Ya	128 MB	1500	33	100%
23	earl	python27-gunicorn	2	128 MB	Tidak	-	1500	34	100%
24	bertha	php55-fpm	2	96 MB	Ya	96 MB	824	30	55%

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
25	fanny	python27-gunicorn	1	64 MB	Ya	128 MB	758	30	51%
26	adriel	nodejs-nodemon	1	128 MB	Tidak	-	701	30	47%
Percobaan ke-3									
27	santino	python27-gunicorn	1	64 MB	Tidak	-	1207	15	80%
28	corine	python27-gunicorn	1	96 MB	Tidak	-	1033	17	69%
29	zander	python27-gunicorn	1	128 MB	Ya	96 MB	1500	18	100%
30	nickolas	python27-gunicorn	1	64 MB	Ya	64 MB	1500	18	100%
31	lesley	python27-gunicorn	1	128 MB	Tidak	-	788	18	53%
32	linwood	php55-fpm	1	128 MB	Tidak	-	1500	18	100%

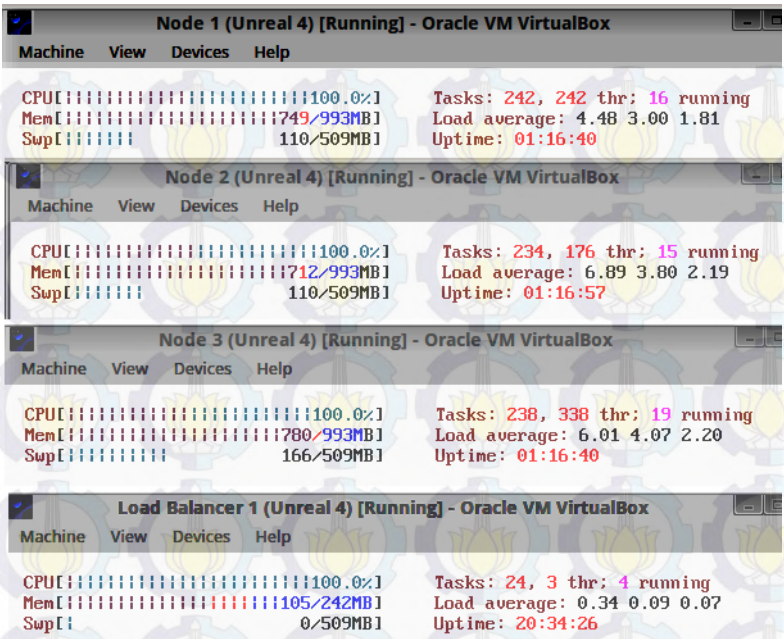
No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
33	kylie	python27-gunicorn	1	64 MB	Tidak	-	1500	18	100%
34	aditya	ruby19-thin	1	128 MB	Tidak	-	1500	18	100%
35	elliot	nodejs-nodemon	2	128 MB	Ya	128 MB	228	23	15%
36	alba	php55-fpm	2	96 MB	Ya	96 MB	95	18	6%
37	elody	ruby19-thin	2	128 MB	Ya	96 MB	1500	18	100%
38	kay	ruby19-thin	2	64 MB	Ya	128 MB	1500	18	100%
39	vince	ruby19-thin	1	64 MB	Tidak	-	1239	18	83%
40	bethany	ruby19-thin	1	128 MB	Ya	64 MB	743	18	50%
41	demarco	php55-fpm	1	64 MB	Ya	64 MB	559	17	37%

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
Percobaan ke-4									
42	charley	python27-gunicorn	1	96 MB	Ya	64 MB	691	27	46%
43	yasmine	python27-gunicorn	1	64 MB	Tidak	-	741	30	49%
44	ethan	nodejs-nodemon	1	96 MB	Ya	96 MB	338	27	23%
45	cullen	php55-fpm	1	64 MB	Tidak	-	542	27	36%
46	caesar	python27-gunicorn	1	128 MB	Tidak	-	678	29	45%
47	jovani	python27-gunicorn	3	64 MB	Tidak	-	1500	33	100%
48	gordon	python27-gunicorn	2	96 MB	Tidak	-	1438	29	96%
49	charlie	nodejs-nodemon	2	96 MB	Tidak	-	1273	28	85%

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
50	marquise	php55-fpm	1	64 MB	Tidak	-	513	27	34%
51	camylle	python27-gunicorn	3	64 MB	Ya	128 MB	1000	26	67%
52	lavon	nodejs-nodemon	1	64 MB	Tidak	-	761	27	51%
53	dusty	python27-gunicorn	2	96 MB	Tidak	-	852	21	57%
54	katelyn	php55-fpm	1	96 MB	Tidak	-	578	27	39%
55	myrl	nodejs-nodemon	2	96 MB	Ya	64 MB	294	24	20%
56	charles	php55-fpm	1	128 MB	Tidak	-	380	19	25%
57	vena	ruby19-thin	1	96 MB	Tidak	-	841	18	56%
58	joe	nodejs-nodemon	1	96 MB	Tidak	-	494	19	33%

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
59	vanessa	ruby19-thin	1	64 MB	Ya	64 MB	653	19	44%
Percobaan ke-5									
60	cole	nodejs-nodemon	1	128 MB	Tidak	-	621	42	41%
61	keyshawn	nodejs-nodemon	1	96 MB	Ya	64 MB	231	51	15%
62	queenie	python27-gunicorn	1	96 MB	Tidak	-	785	38	52%
63	randi	php55-fpm	2	128 MB	Tidak	-	1061	42	71%
64	peter	python27-gunicorn	1	128 MB	Ya	128 MB	758	39	51%
65	elliott	ruby19-thin	3	64 MB	Ya	96 MB	1499	51	100%
66	emilia	ruby19-thin	3	128 MB	Tidak	-	1500	63	100%

No	Nama	Platform	Node	MemApp	DB	MemDb	Uji Performa (Akses HTTP)		
							Sukses	Halaman/s	Keter-sediaan
67	darrion	ruby19-thin	1	128 MB	Tidak	-	897	40	60%
68	hiram	nodejs-nodemon	2	96 MB	Ya	128 MB	409	39	27%
69	brice	ruby19-thin	1	64 MB	Ya	96 MB	724	39	48%
70	tod	ruby19-thin	1	128 MB	Ya	96 MB	736	35	49%
71	elza	php55-fpm	1	64 MB	Ya	64 MB	235	57	16%
72	electa	ruby19-thin	1	96 MB	Ya	64 MB	750	36	50%



Gambar 5.6: Keadaan Node dan Load Balancer Ketika Sedang Uji Performa

Sebagai informasi, akses "Sukses" dihitung dari jumlah permintaan HTTP yang menghasilkan respons 200 (berhasil tanpa galat). Sementara ketersediaan atau *availability* dihitung berdasarkan persentase "Sukses" terhadap jumlah semua permintaan HTTP ke aplikasi (500 x 3 penguji = 1500 permintaan).

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 ren-

dah 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.

Kecepatan akses halaman dalam satuan halaman per detik (halaman/s) cenderung bervariasi untuk setiap percobaan. Pada satu percobaan, kecepatan akses ada di kisaran (40 - 60) sedangkan pada percobaan lain, kecepatan tidak ada yang mencapai lebih dari 30 halaman/s. Kemungkinan penyebab dari adanya variasi kecepatan adalah karena pengujian dilakukan menggunakan komputer fisik terpisah sehingga kemungkinan ada pengaruh keadaan jaringan (misalnya penumpukan paket pada *switch* atau *router*) pada saat uji coba dijalankan.

5.3.3 Implementasi Prinsip Komputasi Awan

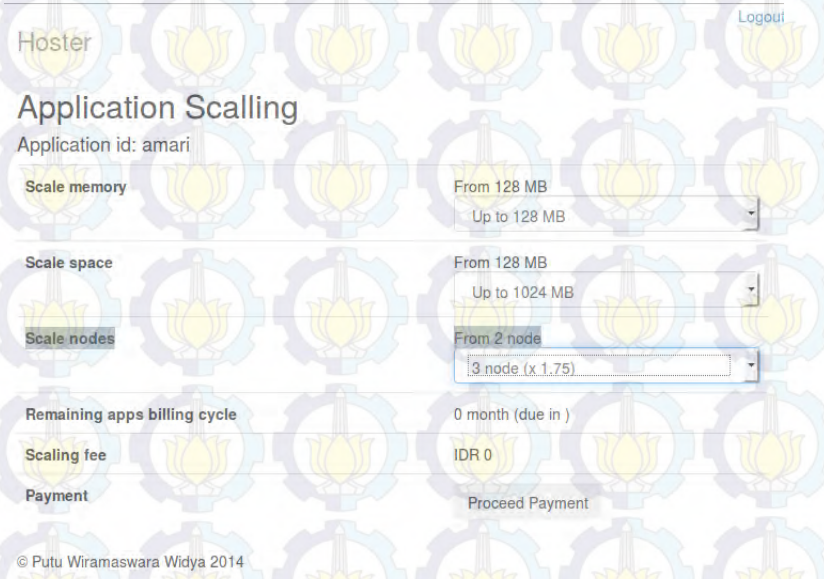
Pada sub-bab berikut, dijelaskan mengenai bagaimana beberapa prinsip komputasi awan yang dipaparkan dalam rumusan masalah dapat diimplementasi di dalam sistem.

5.3.3.1 On-Demand Self-Service

Prinsip **On-Demand Self-Service** yang menurut definisi NIST memungkinkan pelanggan atau penyewa dapat menetapkan kemampuan komputasinya tanpa perlu interaksi manusia ke penyedia layanan, 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 ber-

komunikasi 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 (melalui antarmuka seperti pada Gambar 5.7).



Gambar 5.7: Antarmuka untuk Scaling atau Peningkatan Kapasitas Aplikasi pada Sistem Terimplementasi.

Penagihan adalah salah satu bagian sistem yang bisa dikembangkan lebih lanjut sehingga benar-benar bebas dari interakasi manusia, misalnya dengan menggunakan konsep pembayaran berbasis

kredit atau poin yang bisa ditagihkan setiap kali aplikasi dan basis data miliki penyewa membutuhkan kemampuan sistem lebih lanjut. Jika sistem akan dikembangkan ke arah tersebut, maka perlu adanya komponen evaluator yang bisa menilai apakah suatu aplikasi dari penyewa perlu ditingkatkan kapasitasnya ketika terjadi beban akses, dan kemudian dapat melakukan pengaturan *scaling* secara otomatis seperti pada prinsip **Rapid elasticity**.

5.3.3.2 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 5.8 memperlihatkan banyaknya proses di dalam Docker yang untuk semua aplikasi dan basis data pada sebuah Node.

Banyaknya aplikasi di dalam Node tidak serta merta membuat performa salah satu atau beberapa aplikasi menjadi turun. Sesuai dengan hasil uji performa yang mendapatkan beberapa aplikasi bisa tersedia 100% selama di-*benchmark* bersamaan dengan aplikasi lainnya. Walaupun demikian, perlu adanya tinjauan lebih lanjut mengenai konfigurasi citra cakram Docker dengan pengaturan yang sesuai agar aplikasi dan basis data yang berjalan memiliki indeks ketersediaan semaksimal mungkin.

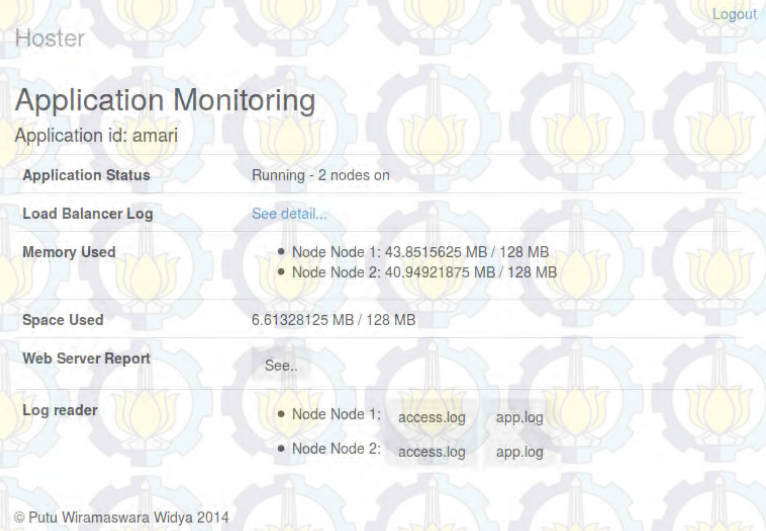
5.3.3.3 Measured Service

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 sendi-

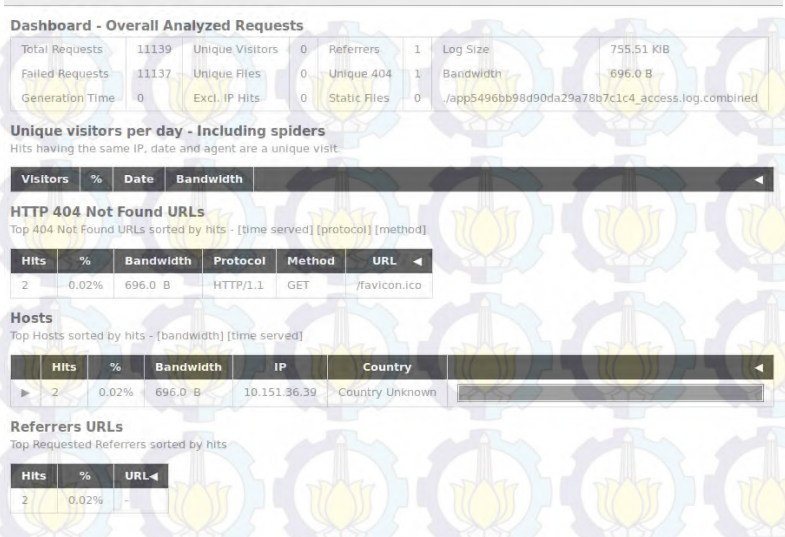
```
root@node1:/opt/tugasakhir# docker ps
CONTAINER ID        IMAGE                                COMMAND             CREATE
D                   STATUS              PORTS
 NAMES
e829cc125586        hoster/python27-gunicorn:latest   "/sbin/my_init"     3 hour
s ago          Up 3 hours          0.0.0.0:50827->22/tcp, 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 5.8: Hasil dari Perintah `docker ps` yang Menampilkan Daftar Kontainer Docker yang Berjalan untuk Mengakomodir Banyaknya Aplikasi dan Basis Data pada Suatu Node

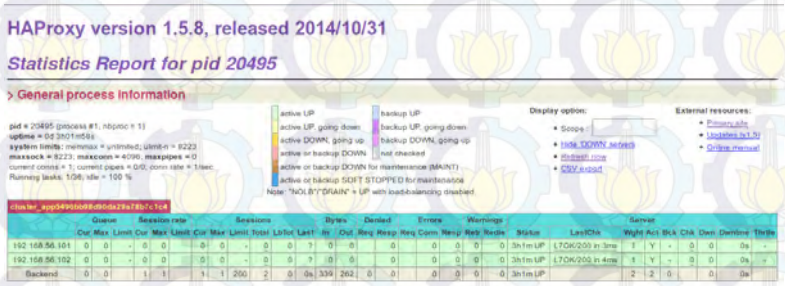
ri dalam bentuk laporan (seperti pada Gambar 5.9, 5.10 dan 5.11) yang dapat memberikan pengetahuan kepada pelanggan mengenai bagaimana aplikasi mereka diakses oleh klien.



Gambar 5.9: Antarmuka Pemantauan Sebuah Aplikasi Secara Umum, Dapat Dilihat Penggunaan Memori dan Cakram Penyimpanan pada Berjalannya Aplikasi Bersangkutan



Gambar 5.10: Antarmuka Pemantauan Rangkuman Akses HTTP pada Aplikasi



Gambar 5.11: Antarmuka Pemantauan Penggunaan Penyeimbang Muat pada Aplikasi (melalui antarmuka `stats` dari HAProxy)

BAB 6

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya terhadap hasil uji coba yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian terhadap sistem lebih lanjut.

6.1 Kesimpulan

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

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 melalui sistem layanan pengelolaan Load Balancer dan Node secara otomatis, sistem pembagian manajemen sumber daya melalui virtualisasi berbasis Docker dan pengukuran layanan berbasis jumlah Node, jumlah memori RAM dan jumlah ruang cakram penyimpanan untuk setiap aplikasi dan basis data yang dimiliki oleh penyewa.

6.2 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.

DAFTAR 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] DewaWeb, **Cloud Hosting for Personal**, [Online], http://www.dewaweb.com/cloud-hosting-for-personal/, diakses tanggal 22 September 2014

[3] DeDoHo, **Cloud Hosting**, [Online], https://www.dedoho.pw/hosting/cloud-hosting/, diakses tanggal 22 September 2014

[4] Cloud Kilat, **Kilat Hosting**, [Online], http://www.cloudkilat.com/harga/kilat-hosting, diakses tanggal 22 September 2014

[5] OpenShift Developer, **OpenShift's Features**, [Online], https://developers.openshift.com/en/overview-platform-features.html, diakses tanggal 22 September 2014

[6] Heroku, **Heroku's Features**, [Online], https://www.heroku.com/features, diakses tanggal 22 September 2014

[7] cPanel, **Full feature list**, [Online], http://cpanel.net/cpanel-whm/full-feature-list/, diakses tanggal 22 September 2014

[8] 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

[9] 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.

[10] Guillermo Rauch, **Smashing Node.js: JavaScript Everywhere**, Wiley, First Edition, 2012

[11] Joyent, inc. ***node*.js**, [Online], http://www.nodejs.org/, Diakses tanggal 2 Desember 2014

[12] 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

[13] Scott Davis, **Mastering MEAN: Introducing to MEAN Stack**, [Online], http://www.ibm.com/developerworks/library/wa-mean1/index.html, diakses tanggal 21 Desember 2014

[14] James Turnbull, **The Docker Book: Containerization is a the new Virtualization**, version 1.3.2, James Turnbull, 2014

[15] Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia, **Virtualization vs Containerization to support PaaS**, 2014 IEEE International Confrence on Cloud Engineering, pp 610-614, 2014

[16] HAProxy, **HAProxy Description**, [Online], http://www.haproxy.org/#desc, diakses tanggal 15 Desember 2014.

[17] 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.

LAMPIRAN A

PERBANDINGAN LAYANAN CLOUD WEB HOSTING

Pada lampiran berikut, dijelaskan mengenai perbandingan antara layanan yang menamakan dirinya sebagai *cloud* web hosting di Indonesia yang semuanya berbasis cPanel pada tabel A.1 sementara perbandingan fitur layanan cloud *hosting* di luar negeri antara yang berbasis cPanel, Heroku dan OpenShift pada tabel A.2.

Tabel A.1: Layanan yang Dipasarkan sebagai Cloud Web Hosting di Indonesia

No	Nama	URL	Panel	Fitur
1	Dewa Web [2]	`http://dewaweb.com`	cPanel	Unlimited Bandwidth, CloudFare CDN, Sumberdaya tergaransi, Autobackup
2	DeDoHo [3]	`http://dedoho.pw`	cPanel	Unlimited Bandwidth, CloudFare CDN, Autobackup
3	CloudKilat [4]	`http://cloudkilat.com`	cPanel	Unlimited Bandwidth, CloudFare CDN

Tabel A.2: Perbandingan Fitur antara Cloud Hosting berbasis cPanel dengan Layanan OpenShift dan Heroku

No	Fitur	cPanel-based [7]	OpenShift [5]	Heroku [6]
1	Kemampuan *scaling*	Menghubungi admin, tidak *multi-node*	Otomatis sesuai trafik atau manual oleh pengguna	Manual oleh pengguna
2	Dukungan PHP	Ada, kebanyakan tidak menyediakan variasi versi	Ada, bisa memilih versi	Ada, bisa memilih versi
3	Dukungan Node.js	Tidak ada	Ada	Ada
4	Dukungan Ruby	Tidak ada	Ada	Ada
5	Dukungan Python	Tidak ada	Ada	Ada
6	Basis Data MySQL	Ada, satu *server* dibagi banyak basis data	Ada, satu *server* untuk satu aplikasi	Ada, satu *server* untuk satu aplikasi
7	Basis Data MongoDB	Tidak ada	Ada	Ada
8	Akses Berkas Aplikasi	File manager / FTP	Git	Git/Mercurial
9	Dukungan Websockets	Tidak ada	Ada	Ada

No	Fitur	cPanel-based [7]	OpenShift [5]	Heroku [6]
10	Sistem penagihan	Bayar diawal, tergantun kapasitas	Gratis, Bronze, Silver	Sesuai kemampuan resource (dyno)
11	Biaya	Mulai Rp. 10.000 per bulan	Paket bronze mulai 0 USD	Mulai 34.50 USD per bulan
12	Membangun Layanan Serupa	Bisa dengan bayar lisensi	OpenShift Origin, namun untuk kebutuhan *private* cloud bukan *web hosting*	Tidak Bisa

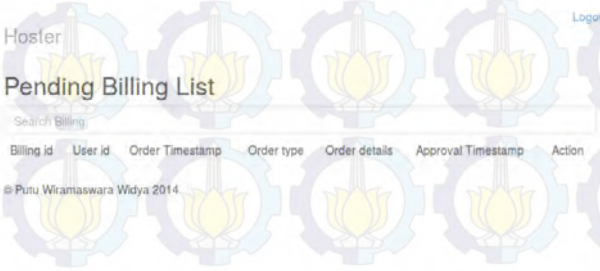
Halaman ini sengaja dikosongkan

LAMPIRAN B

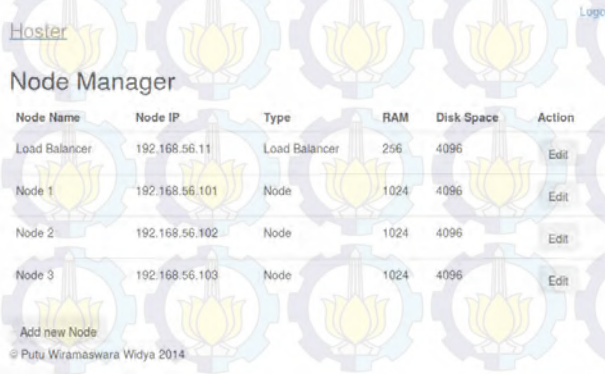
TANGKAPAN LAYAR PANEL

Pada lampiran berikut, ditampilkan mengenai tangkapan layar aplikasi *frontend* Manager yang diakses melalui aplikasi peramban Web.

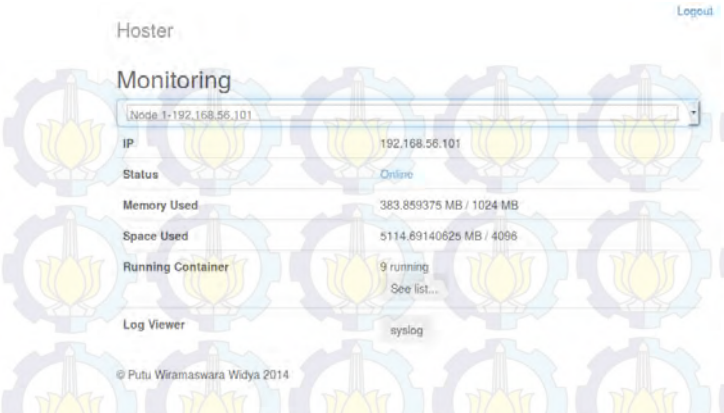
B.1 Antarmuka Admin



Gambar B.1: Antarmuka Daftar Penagihan dari Penyewa

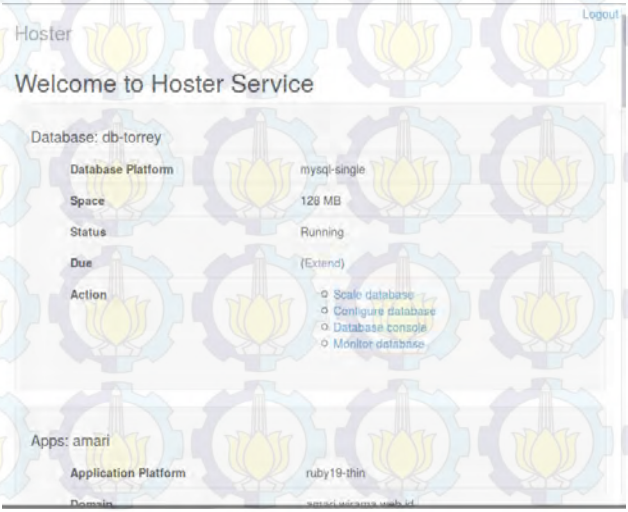


Gambar B.2: Antarmuka Kelola Daftar Node dan Load Balancer

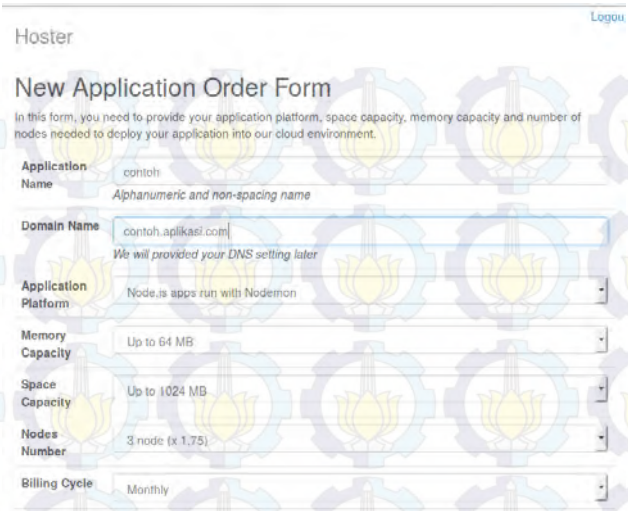


Gambar B.3: Antarmuka Pemantauan Keadaan Node Secara Umum

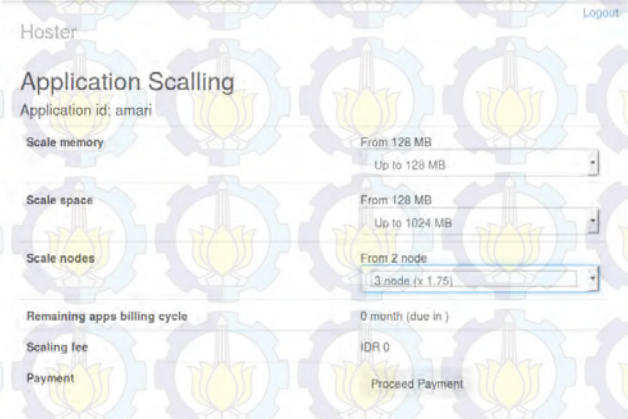
B.2 Antarmuka Penyewa



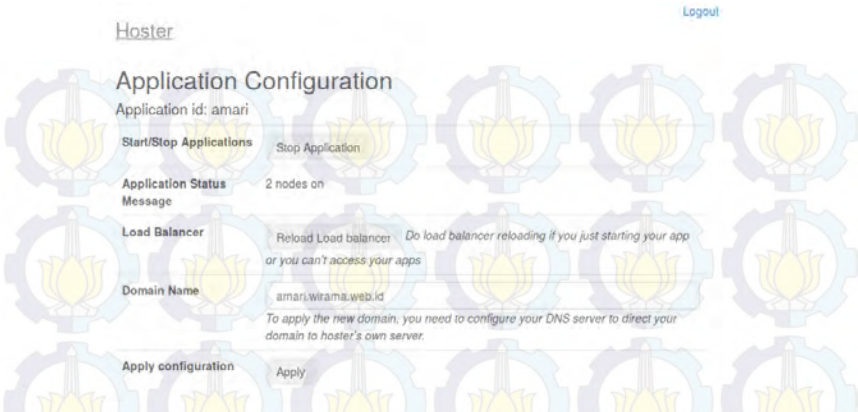
Gambar B.4: Antarmuka Dasbor Pengguna



Gambar B.5: Antarmuka Pemesanan Aplikasi/Basis Data Baru

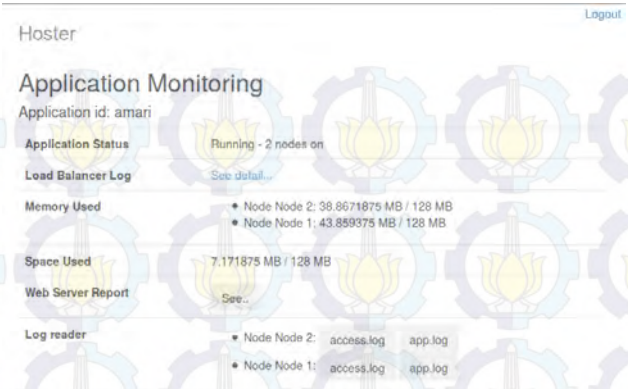


Gambar B.6: Antarmuka Pemesanan Peningkatan Kapasitas Aplikasi/Basis Data

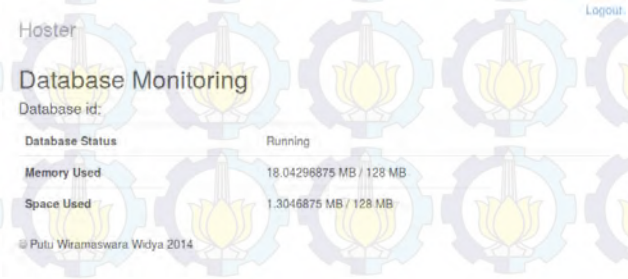


Gambar B.7: Antarmuka Pengaturan Aplikasi

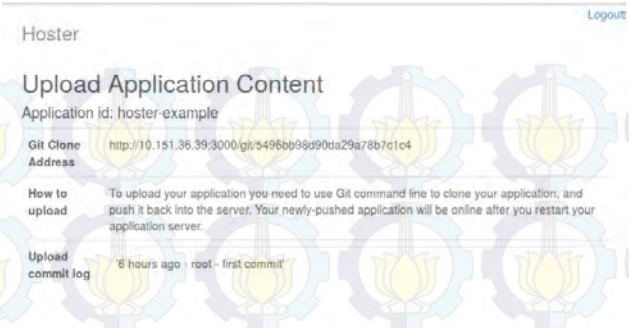
Gambar B.8: Antarmuka Pengaturan Basis Data



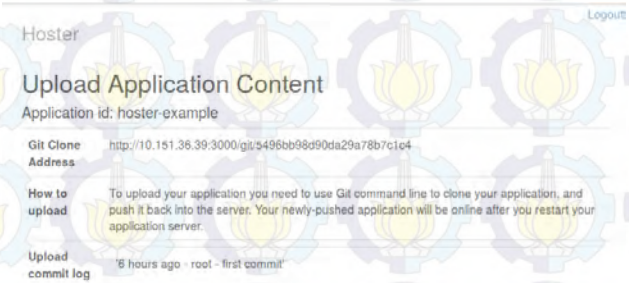
Gambar B.9: Antarmuka Pemantauan Aplikasi



Gambar B.10: Antarmuka Pemantauan Basis Data



Gambar B.11: Antarmuka Pemantauan Unggahan Aplikasi melalui Git



Gambar B.12: Antarmuka Pengelolaan Basis Data Melalui Konsol MySQL

BIODATA PENULIS

Putu Wiramaswara Widya, biasa dipanggil Wira lahir di Denpasar tanggal 28 September 1993. Merupakan anak sulung dari dua bersaudara, penulis menyelesaikan hampir 17 tahun masa hidupnya di Kabupaten Klungkung, Provinsi Bali. Sejak kecil, penulis sudah memiliki ketertarikan dalam bidang komputer terutama dalam bidang sistem operasi berbasis Linux dan jaringan komputer. Penulis pernah membantu melakukan pembuatan program berbasis Visual Basic 6.0 pada saat masih mengenyam pendidikan di Sekolah Dasar (SD) dan melakukan instalasi sistem operasi Mandrakelinux 10.0 pada tahun 2004. Penulis aktif dalam organisasi daring sejak tahun 2007 melalui berbagai milis seperti `id-ubuntu`, `id-slackware` dan `BlankOn-Dev`. Pada masa Sekolah Menengah Atas, penulis sempat menjadi pengembang distribusi Linux lokal BlankOn Linux (`http://blankonlinux.or.id`) dan menjadi penanggungjawab pengembangan sistem metode input dan fonta pintar Unicode untuk aksara Bali. Penulis pernah meraih medali emas pada ajang Olimpiade Penelitian Siswa Indonesa (OPSI) 2010 dalam bidang sains terapan. Saat ini, penulis masih terus berkecimpung di bidang pengembangan dan penggunaan Linux serta perangkat lunak sumber terbuka (Free Open Source Software, FOSS) dan sedang mendalami berbagai kerangka kerja bahasa pemrograman untuk pengembangan aplikasi Web dan jaringan seperti Python dan Node.js. Penulis memiliki blog pada alamat `http://wirama.web.id/blog` (dalam bahasa Inggris) dan dapat dihubungi melalui surel di `putu@wirama.web.id` atau `initrunlevel0@gmail.com`.