TUGAS AKHIR - KI141502

RANCANG BANGUN MANAJEMEN ALOKASI VIRTUAL MACHINE DALAM LINGKUNGAN HYPERVISOR YANG HETEROGEN

FATHONI ADI KURNIAWAN
NRP 5114 100 020

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

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

RANCANG BANGUN MANAJEMEN ALOKASI VIRTUAL MACHINE DALAM LINGKUNGAN HYPERVISOR YANG HETEROGEN

FATHONI ADI KURNIAWAN
NRP 5114 100 020

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

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMENT INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

DESIGN OF VIRTUAL MACHINE ALLOCATION MANAGEMENT IN HETEROGENEOUS HYPERVISOR ENVIRONMENT

FATHONI ADI KURNIAWAN
NRP 5114 100 020

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

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D

INFORMATICS DEPARTMENT
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN MANAJEMEN ALOKASI VIRTUAL MACHINE DALAM LINGKUNGAN HYPERVISOR YANG HETEROGEN

TUGAS AKHIR
Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur Jaringan dan Komputer
Program Studi S1 Jurusan Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
FATHONI ADI KURNIAWAN
NRP: 5114 100 020

Disetujui oleh Dosen Pembimbing Tugas Akhir :

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

Bagus Jati Santoso, S.Kom., Ph.D .....
NIP: 051100116 (Pembimbing 2)

SURABAYA
Juni 2018

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN MANAJEMEN ALOKASI VIRTUAL MACHINE DALAM LINGKUNGAN HYPERVISOR YANG HETEROGEN

Nama	:	**FATHONI ADI KURNIAWAN**
NRP	:	**5114 100 020**
Jurusan	:	**Informatika FTIK**
Pembimbing I	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D**
Pembimbing II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstrak

Saat ini, dengan didukung oleh konsep SaaS (Software as a Service), aplikasi web berkembang dengan pesat. Para penyedia layanan aplikasi web berlomba-lomba memberikan pelayanan yang terbaik, seperti menjaga QoS (Quality of Service) sesuai dengan perjanjian yang tertuang dalam SLA (Service Level Agreement). Hal tersebut dikarenakan permintaan akses ke suatu aplikasi web biasanya meningkat dengan seiring berjalannya waktu. Keramaian akses sesaat menjadi hal yang umum dalam aplikasi web saat ini. Saat hal tersebut terjadi, aplikasi web akan di akses lebih banyak dari kebiasaan. Jika aplikasi web tersebut tidak menyediakan kemampuan untuk menangani hal tersebut, bisa menyebabkan aplikasi web tidak dapat berjalan dengan semestinya yang sangat merugikan pengguna.

Elastic cloud merupakan salah satu bagian dari komputasi awan yang sedang populer, dimana banyak riset dan penelitian yang berfokus di bidang ini. Elastic cloud bisa digunakan untuk menyelesaikan permasalah di atas. Lalu sebuah perangkat lunak bernama Docker dapat dapat diterapkan untuk mendukung elastic cloud.

Dalam tugas akhir ini akan dibuat sebuah rancangan sistem

yang memungkinkan aplikasi web berjalan di atas Docker. Sistem ini bisa beradaptasi sesuai dengan kebutuhan dari aplikasi yang sedang berjalan. Jika aplikasi membutuhkan sumber daya tambahan, sistem akan menyediakan sumber daya berupa suatu container baru secara otomatis dan juga akan mengurangi penggunaan sumber daya jika aplikasi sedang tidak membutuhkannya. Dari hasil uji coba, sistem dapat menangani sampai dengan 57.750 request dengan error request yang terjadi sebesar 7.83%.

***Kata-Kunci**: aplikasi web, autoscale, docker, elastic cloud*

DESIGN OF VIRTUAL MACHINE ALLOCATION MANAGEMENT IN HETEROGENEOUS HYPERVISOR ENVIRONMENT

Name	:	**FATHONI ADI KURNIAWAN**
NRP	:	**5114 100 020**
Major	:	**Informatics FTIK**
Supervisor I	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D**
Supervisor II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstract

Nowdays, with the concept of SaaS (Software as a Service), web applications have developed a lot. Web service providers are competing to provide the best service, such as QoS (Quality of Service) requirements specified in the SLA (Service Level Agreement). The load of web applications usually very drastically along with time. Flash crowds are also very common in today's web applications world. When flash crowds happens, the web application will be accessed more than usual. If the web applications does not provide the ability to do so, it can make the web application not work properly which is very disadvantegeous to the users.

Elastic cloud is one of the most popular part of cloud computing, with much researchs in this subject. Elastic clouds can be used to solve the above problems. Then a Docker can be applied to support the elastic cloud.

In this final task will be made an application system that allows web applications running on top of Docker. This system can adjust according to the needs of the running applications. If the application requires additional resources, the system will automatically supply the resources of a new container and will

also reduce resource usage if the application is not needing it. From the test results, the system can handle up to 57,750 requests and error ratio of 7.83%.

***Keywords**: autoscale, docker, elastic cloud, web application*

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang Bangun Manajemen Alokasi *Virtual Machine* dalam Lingkungan *Hypervisor* yang Heterogen**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Departemen Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Keluarga penulis yang selalu menyemangati.
3. Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D selaku pembimbing I yang telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Bapak Bagus Jati Santoso, S.Kom., Ph.D selaku pembimbing II yang juga telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
5. Teman-teman *Administrator* laboratorium AJK.
6. Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Departemen Informatika ITS pada masa pengerjaan Tugas Akhir, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku

koordinator TA dan segenap dosen Departemen Informatika yang telah memberikan ilmu dan pengalamannya.

7. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2018

Fathoni Adi K

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir dan sistematika penulisan.

1.1 Latar Belakang

Virtual Machine merupakan teknik virtualisasi yang menyajikan perangkat keras yang dapat menjalankan perangkat lunak seperti perangkat keras fisik. Penyedia layanan *Virtual Machine* biasa disebut dengan *Hypervisor*. *Hypervisor* menangani manajemen *Virtual Machine* pada sebuah *host*.

Saat ini penggunaan *Hypervisor* dalam dunia teknologi sangat banyak dilakukan. Karena sangat banyaknya penggunaan Hypervisor, maka dari itu banyak penyedia-penyedia *Hypervisor* contohnya Vmware, Proxmox, Xen, Qemu dll. Keragaman *Hypervisor* menyebabkan perbedaan cara pengoperasian. Hal ini menyebabkan sulitnya alokasi *Virtual Machine*. Masalah tersebut juga dialami oleh DPTSI ITS. Ketika pengguna membutuhkan *Virtual Machine* untuk kerperluan pengembangan aplikasi maupun server database, pengguna akan kebingungan untuk melakukan alokasi *Virtual Machine*. Pada akhirnya pengguna yang membutuhkan *Virtual Machine* akan menghubungi *System Administrator* yang tahu tentang pengorperasian Hypervisor tertentu.

Oleh karena itu dibutuhkan cara untuk mengelola alokasi *Virtual Machine* pada *Hypervisor* yang berbeda. Salah satunya dengan membuat sebuah *Middleware* yang bertugas untuk menjembatani cara penggunaan *Hypervisor* yang berbeda. *Middleware* akan diakses melalui *interface* yang disediakan untuk pengguna untuk manjemen alokasi *Virtual Machine*.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana cara membuat *Middleware* berbasis *REST API* yang digunakan untuk menjembatani penggunaan *Hypervisor* yang berbeda?
2. Bagaimana cara mengimplementasikan *Middleware* berbasis *REST API* melalui *interface* web dan *command line*?
3. Bagaimana cara menentukan *Host* yang tersedia untuk alokasi *Virtual Machine* menggunakan algoritma *Decision Making (Analytical Hierarchy Process)*

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. *Hypervisor* yang didukung adalah Proxmox dan Vmware.
2. OS yang disediakan untuk *Virtual Machine* adalah template Ubuntu dan Debian.
3. Uji coba aplikasi akan menggunakan *REST API*.

1.4 Tujuan

Tujuan pembuatan tugas akhir ini antara lain:

1. Membuat sebuah *Middleware* yang digunakan untuk menjembatani penggunaan *Hypervisor* yang berbeda.
2. Mengimplementasikan metode pengambilan keputusan untuk pendistribusian kontainer yang efisien berdasarkan penggunaan RAM, CPU dan Penyimpanan File pada server

1.5 Manfaat

Manfaat dari pembuatan tugas akhir ini adalah mempermudah pengelolaan *Virtual Machine* dalam lingkungan *Hypervisor* yang beragam (heterogen) sehingga pengguna tidak perlu tahu bagaimana cara menggunakan masing-masing *Hypervisor*.

(Halaman ini sengaja dikosongkan)

BAB II

TINJAUAN PUSTAKA

2.1 Virtualisasi

Virtualisasi merupakan komponen terpenting dalam komputasi awan dengan memisahkan perangkat keras dengan sistem operasi yang berjalan. Virtualisasi memiliki kemampuan untuk membagi sumber daya fisik yang ada untuk dijadikan sumber daya *virtual* dan dapat menjadikan berbagai sumber daya fisik yang ada menjadi satu sumber daya *virtual*. Virtualisasi membawa banyak perubahan pada perusahaan IT saat ini.

Virtualisasi dan *multitasking* pada sistem operasi memiliki kemampuan untuk mengizinkan pemusatan berbagai server *virtual* pada sebuah komputer fisik. Ketika sebuah kelompok ingin mengerjakan sebuah pekerjaan tertentu pada dua atau lebih server dan salah satu server gagal karena sumber daya habis terpakai, virtualisasi dapat memindahkan pekerjaan dan server tersebut pada komputer fisik yang lain. Hal tersebut merupakan salah satu keuntungan menggunakan virtualisasi. Selain itu virtualisasi dapat menghemat energi yang dipakai dan biaya pembelian komputer fisik. Sehingga virtualisasi dapat meningkatkan keuntungan perusahaan[1].

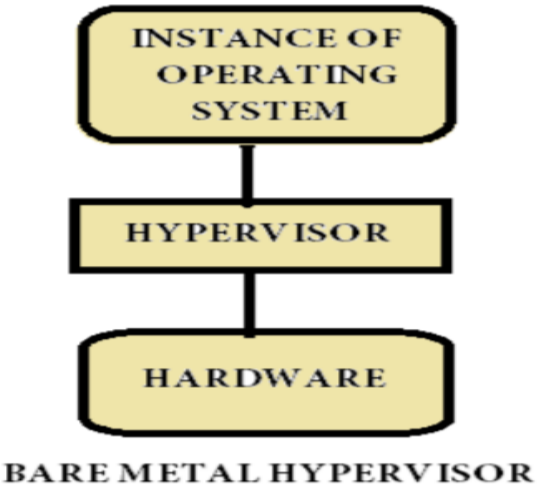
2.2 *Hypervisor*

Pada virtualisasi, terdapat sebuah *layer* yang berada diantara perangkat keras dan *virtual machine*. *Layer* tersebut disebut *Hypervisor*. *Hypervisor* menyediakan standarisasi *CPU*, *memory* dan *storage* untuk *virtual machine*. *Hypervisor* memiliki dua jenis yaitu, *Bare-metal Hypervisor* dan *Hosted Hypervisor*[1].

2.2.1 *Bare-metal Hypervisor*

Jenis *hypervisor* ini dilakukan instalasi pada pada perangkat keras x86 secara langsung. Setelah melakukan instalasi

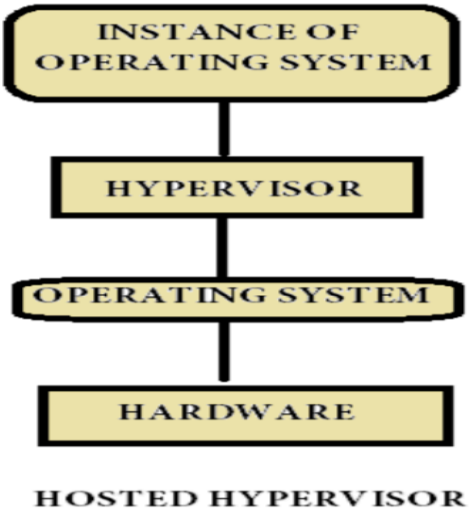
hypervisor, kita dapat melakukan instalasi sistem operasi pada (*virtual machine*) atau yang sering disebut *instance*. Jenis *hypervisor* ini lebih efisien dibanding *Hosted hypervisor*.



Gambar 2.1: Arsitektur *Bare-metal Hypervisor*

2.2.2 *Hosted Hypervisor*

Jenis *hypervisor* ini dilakukan instalasi pada pada sistem operasi yang sudah terinstalasi pada perangkat keras sebelumnya. *Instance* dibuat setelah melakukan instalasi *hypervisor*.



Gambar 2.2: Arsitektur *Hosted Hypervisor*

2.3 Python

Python adalah bahasa pemrograman interpretatif, interaktfi dan berorientasi objek. Python menggabungkan modul, pengecualian, penulisan secara dinamis, tipe data dinamis yang sangat tinggi dan kelas. Python memiliki antarmuka ke banyak *system call* dan pusataka diberbagai sistem dan dapat diperluas ke bahasa pemrograman C atau C++. Python dapat berjalan pada berbagai sistem operasi seperti Unix, Linux, Mac Os dan Windows.

Python adalah bahasa pemrograman tingkat tinggi yang dapat diterapkan pada berbagai masalah. Bahasa ini dilengkapi pustaka yang besar untuk melakukan pemrosesan *string*, protokol internet, rekayasa perangkat lunak dan antarmuka sistem operasi[2].

2.4 Flask

Flask adalah kerangka aplikasi web Python yang ringan. Flask dirancang untuk memulai membuat web dengan cepat dan mudah, dengan kemampuan untuk membuat aplikasi web sampai tingkat yang rumit. Flask dibuat dengan terintegrasi dengan modul Werkzeug dan Jinja. Flask termasuk salah satu kerangka aplikasi web Python yang populer.

Flask didesain tidak memiliki depedensi dan tata letak kerangka aplikasi, dengan demikian pengembang memiliki kebebasan untuk mengatur kerangka aplikasinya sendiri serta menambahkan modul yang diperlukan sesuai kebutuhan. Flask memiliki berbagai ekstensi yang dikembangkan oleh komunitas sehingga dapat menambahkan berbagai fungsi dengan mudah[3].

2.5 Python Celery

Celery adalah modul Python yang berguna untuk antrian pekerjaan bersifat asinkron yang berdasarkan pengiriman pesan secara distribusi. Celery berfokus pada operasi *real-time* namun juga mendukung penjadwalan. Unit yang dieksekusi disebut dengan *task* yang dijalankan bersamaan pada satu server atau lebih yang menggunakan mekanisme *multiprocessing*.

Celery dapat diintegrasikan pada kerangka kerja web dengan mudah. Selain itu meskipun Celery ditulis dengan Python, tetapi protokolnya dapat diimplementasikan dalam bahasa apa pun. Ini juga dapat beroperasi dengan bahasa lain menggunakan *webhooks*[4].

2.6 Pyvmomi

Pyvmomi adalah Python SDK untuk Vmware Sphere yang digunakan untuk mengatur Vmware ESX, ESXI dan Vcenter.

Pyvmomi dikembangkan langsung oleh Vmware untuk mempermudah pengaturan secara otomatis[5].

2.7 Proxmoxer

Proxmoxer adalah Python *wraper* untuk mengakses API Proxmox versi 2 melalui protokol HTTPS dan SSH. Modul ini dikembangkan oleh komunitas[6].

2.8 PHP

PHP adalah bahasa pemrograman *scripting* yang memiliki lisensi terbuka yang cocok untuk membuat aplikasi web dan dapat dimasukkan pada HTML. PHP dapat PHP dapat dijalankan pada sistem operasi Unix, Linux, Windows dan Mac Os. Untuk saat ini, PHP saat ini didukung berbagai web server, contohnya Apache, IIS dan lainnya. PHP juga mendukung pemrograman prosedural, pemrograman berbasis objek dan pengabungan dari keduanya. Selain menampilkan HTML, PHP juga dapat menampilkan gambar, PDF dan Flash yang dihasilkan secara cepat. PHP juga didukung oleh berbagai macam database seperti Mysql. Untuk berkomunikasi dengan servis yang lain, PHP mendukung berbagai protokol seperti LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM dan *raw socket*[7].

2.9 Redis

Redis adalah perangkat lunak terbuka penyimpanan data dengan lisensi BSD yang digunakan sebagai wadah untuk menyimpan struktur data dalam *memory* yang digunakan sebagai basis data, cache dan message broker. Redis mendukung penyimpanan dengan berbagai tipe data seperti *strings*, *hashes*, *lists*, *sets*, *sorted sets*, *bitmaps*, *hpreloglogs* dan *geospatial*

indexes. Selain itu, salah satu penggunaan Redis yang umum digunakan adalah sebagai *task queue*.

Dalam hal meningkatkan kinerjanya, Redis bekerja pada *in-memory dataset*. Data yang dikelola oleh Redis berada dalam *memory* sehingga proses membaca dan menulisnya akan cepat dan efisien, selain itu data tersebut bisa dijadikan *persistent* dengan menyimpannya ke dalam *disk*[8].

2.10 MySQL

MySQL adalah merupakan salah basis data resional terbuka yang awalnya dikembangkan oleh MySQL AB. Mulai tahun 2009, setelah Oracle Corporation mengakusisi Sun Microsystems, MySQL dikembangkan dan distribusikan oleh Oracle Corporation. MySQL tersedia sebagai basis data gratis di bawah lesensi *GNU General Public License* (GPL), tetapi juga tersedia lisensi komersial.

MySQL bekerja pada banyak *platform*, seperti Compaq Tru64, DEC OSF, FreeBSD, IBM AIX, HP-UX, Linux, Mac OS X, Novell NetWare, OpenBSD, QNX, SCO, SGI IRIX, Solaris (versions 8, 9 and 10) dan Microsoft Windows. MySQL juga menyediakan *source code* apabila MySQL dalam bentuk *binaries* tidak tersedia pada platform yang digunakan. MySQL menyediakan *API* untuk berbagai bahasa pemrograman seperti C, C++, Java, Perl, PHP, Ruby, Tcl dan lainnya.

MySQL juga menawarkan banyak jenis mekanisme untuk mengelola data, yang dikenal sebagai *storage engines*. MySQL telah lama mendukung beberapa *storage engines*, yaitu MyISAM (standar umum pada semua sistem operasi kecuali Windows), MEMORY (sebelumnya dikenal sebagai HEAP), InnoDB (standar umum pada Windows) dan MERGE. Versi 5 menambahkan mesin ARCHIVE, BLACKHOLE, CSV, FEDERATED dan CONTOH. Baru-baru ini, MySQL telah

merilis versi alpha dari Falcon, sebuah mesin penyimpanan berkinerja tinggi yang ditujukan untuk penyebaran skala besar pada sistem *multi-threaded* / *multi-core*[9].

2.11 Algoritma *Analytical Hierarchy Process*

Analytical Hierarchy Process atau yang sering disebut AHP adalah teknik terstruktur untuk menangani keputusan yang kompleks berdasarkan matematika dan psikologi. AHP dikembangkan oleh Thomas L. Saaty pada tahun 1970an dan telah dipelajari dan disempurnakan secara intensif sejak saat itu. AHP menyediakan kerangka kerja yang komprehensif dan rasional untuk menyusun suatu masalah keputusan, untuk mewakili dan mengkuantifikasi elemen-elemennya, untuk menghubungkan elemen-elemen tersebut dengan tujuan keseluruhan dan untuk mengevaluasi solusi alternatif. AHP telah digunakan diseluruh dunia dalam bidang pemerintahan, bisnis, industri, kesehatan dan pendidikan. Awalnya prioritas ditetapkan sesuai dengan kepentingan untuk mencapai tujuan, setelah prioritas tersebut diturunkan untuk kinerja alternatif pada setiap kriteria, prioritas tersebut diturunkan berdasarkan penilaian berpasangan menggunakan penilaian, atau jatah pengukuran dari skala jika ada. AHP telah digunakan di banyak bidang karena kemampuan untuk menentukan peringkat pilihan sesuai dengan keefektifannya dalam mencapai tujuan yang bertentangan. Penelitian telah berhasil menggunakan AHP dalam memilih satu alternatif dari banyak alokasi sumber daya, peramalan, manajemen kualitas total, rekayasa ulang proses bisnis, penyebaran fungsi kualitas dan nilai skor yang berimbang. AHP adalah metode yang lebih baik, di mana parameter adalah kategori ke dalam sub parameter[10]. Skala umum pada AHP [11] ditunjukkan pada Tabel 2.1

Tabel 2.1: Daftar Skala Prioritas pada AHP

Tingkat Kepentingan	**Definisi**	**Penjelasan**
1	Sama Penting	2 faktor berkontribusi senilai terhadap objektif yang ada.
3	Sedikit Lebih Penting	salah satu faktor lebih penting sedikit dibandingkan faktor yang lain.
5	Lebih Penting	salah satu faktor lebih penting dibandingkan faktor yang lain.
7	Sangat Lebih Penting	salah satu faktor sangat lebih penting dibandingkan faktor yang lain.
9	Benar-benar Lebih Penting	Bukti yang mendukung salah satu faktor dari yang lain telah mencapai kemungkinan yang tertinggi.
2,4,6,8	Nilai Tengah	Nilai saat dimana dibutuhkan kompromi.

Untuk membuat keputusan dengan cara yang terorganisasi untuk menghasilkan prioritas, kita perlu menguraikan keputusan menjadi langkah-langkah berikut[12]:

1. Pengaturan Masalah dan Pemilihan Kriteria.
   Langkah pertama adalah menguraikan masalah pengambilan keputusan menjadi bagian-bagian

penyusunnya. Dalam bentuk yang paling sederhana.

2. Menetapkan kriteria prioritas dengan perbandingan berpasangan (pembobotan).
3. Untuk setiap pasangan kriteria, pengambil keputusan diperlukan untuk menjawab pertanyaan seperti ”Seberapa penting A ke B?” untuk menilai prioritas ”relatif” dari setiap kriteria. Tugas ini dilakukan dengan menetapkan bobot antara 1 dan 9 seperti yang ditunjukkan pada Tabel 2.1 terhadap kriteria yang lebih penting, sementara nilai timbal balik dari nilai ini akan diberikan kepada pasangan kriteria. Pembobotan ini kemudian akan dinormalisasi untuk menambah berat badan untuk setiap kriteria yang disebut Option Performance Matrix (OPM).
4. Perbandingan berpasangan dari pilihan pada setiap kriteria (penilaian).
5. Untuk setiap pasangan dalam setiap kriteria, pilihan yang lebih baik akan diberi nilai antara 1 dan 9, sementara pasangan opsi lainnya akan diberi nilai yang sama dengan nilai timbal balik dari nilai ini. Setiap nilai akan menunjukkan seberapa baik ”A” opsi untuk kriteria ”B”.
6. Terdapat tiga langkah untuk menghitung Weight of Criteria, yakni:
   (a) Hitung n *root of product*
   (b) Priority Vector = nilai dari *n root of product* / jumlah dari *3rd root of product*
   (c) Hitung jumlah nilai dari setiap kolom
7. Dapatkan skor keseluruhan untuk setiap opsi
   Pada langkah terakhir, nilai setiap opsi digabungkan dengan bobot kriteria untuk menghasilkan nilai keseluruhan untuk setiap opsi. Sejauh mana pilihan tersebut memenuhi kriteria akan diukur berdasarkan seberapa penting kriteria tersebut. Ini dilakukan dengan rumus:

$$\sum (\text{kriteria pembobotan} * \text{bobot OPM})$$

(Halaman ini sengaja dikosongkan)

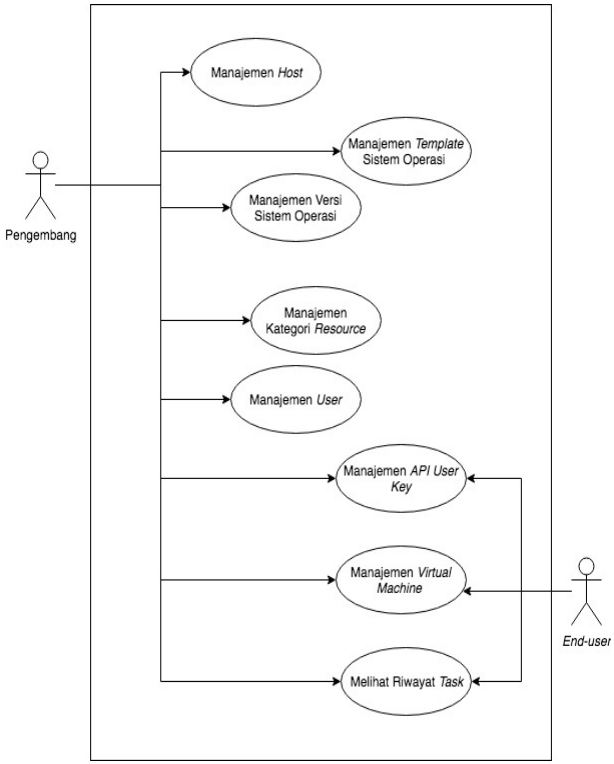
BAB III

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan sistem.

3.1 Kasus Penggunaan

Terdapat dua aktor dalam sistem ini, yaitu pengembang (*administrator*) dan *end-user* (pengguna) dari aplikasi web yang dikelola oleh sistem. Diagram kasus penggunaan digambarkan pada Gambar 3.1.



Gambar 3.1: Diagram Kasus Penggunaan

Diagram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.1.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-0001	Manajemen *Virtual Machine*.	Pengembang dan *end-user* dapat mangatur *virtual machine*.
UC-0002	Manajemen *API Secret Key*.	Pengembang dan *end-user* dapat mengatur *secret key* untuk akses dari aplikasi lain.
UC-0003	Melihat Riwayat *Task*.	Pengembang dan *end-user* dapat melihat *task* yang sedang berjalan atau *task* yang sudah dikerjakan.
UC-0004	Manajemen *Host*.	Pengembang dapat mengatur *host hypervisor* yang tersedia untuk menunjang sistem.
UC-0005	Manajemen *Template* Sistem Operasi.	Pengembang dapat mengatur template sistem operasi yang akan digunakan untuk pembuatan *virtual machine*.
UC-0006	Manajemen Versi Sistem Operasi.	Pengembang dapat mengatur versi sistem operasi.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-0007	Manajemen Kategori *Resource*.	Pengembang dapat mengatur tipe *resource* untuk pembuatan *virtual machine*.
UC-0008	Manajemen *User*.	Pengembang dapat mengatur *user* yang dapat mengakses sistem.

3.2 Arsitektur Sistem

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

3.2.1 Desain Umum Sistem

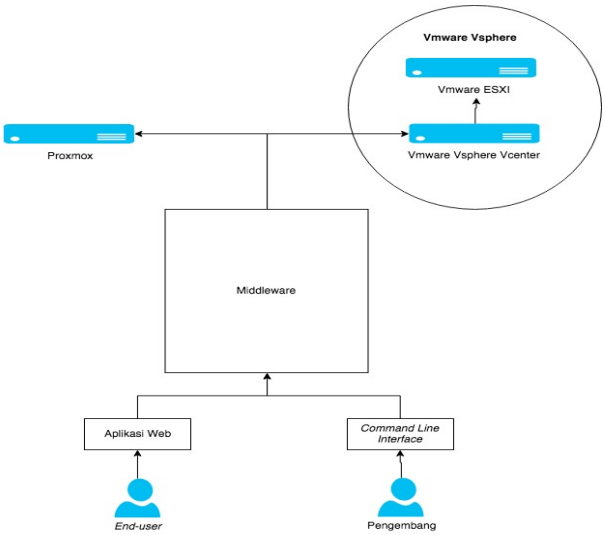
Sistem yang akan dibuat yaitu sistem yang dapat melakukan manajemen alokasi *virtual machine* pada *hypervisor* dalam lingkungan yang heterogen. Sistem yang dikembangkan oleh penulis mendukung alokasi *virtual machine* pada *hypervisor Proxmox* dan *Vmware Vsphere*. Pada saat alokasi *virtual machine* baru, sistem akan memilih server terbaik dengan memperhitungkan presentase ketersediaan sumber daya CPU, memori dan *Storage* dengan menggunakan algoritma *Analytic Hierarchy Process (AHP)*. Setelah menentukan *server* terbaik, sistem akan mengirimkan perintah-perintah alokasi *virtual machine* baru sesuai dengan *hypervisor* yang terinstall pada *server*.

Setiap permintaan alokasi maupun manajemen *virtual machine* akan ditampung pada *queue* terlebih dahulu.

Permintaan-permintaan yang ditampung pada *queue* tersebut akan dikerjakan oleh *worker* yang tersedia.

Sistem ini akan digunakan oleh pengguna, yaitu *end-user* dari aplikasi yang mana hanya bisa melakukan manajemen *manajemen virtual machine*, membuat *secret key* dan melihat riwayat *task*. Selain itu juga digunakan oleh pengembang, yaitu orang mengelola aplikasi. Pengembang dapat melakukan manajemen *host* atau *server*, manejemen *template* sistem operasi, manajemen pengguna *(user)* manajemen kategori *resource*, manajemen versi sistem operasi dan dapat melakukan perkerjaan yang dilakukan oleh pengguna *end-user*.

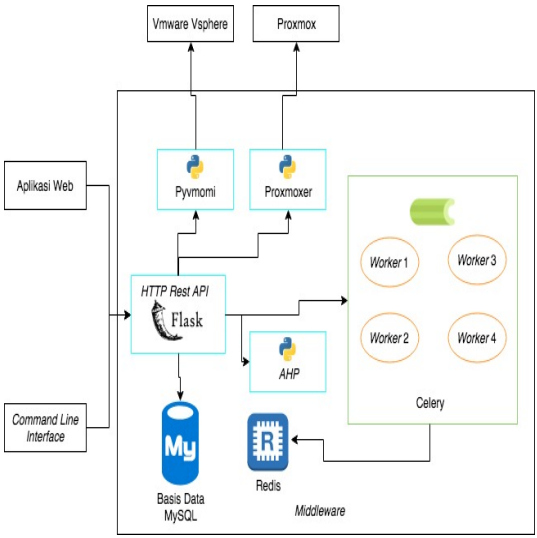
Sistem ini dapat diakses oleh pengguna maupun pengembang melalui aplikasi web dan *command line interface (CLI)*. Aplikasi web dan *command line interface (CLI)* akan meneruskan permintaan pada *middleware* yang merupakan inti penting pada sistem ini. *Middleware* berupa *webservice REST API*. Pada saat alokasi *virtual machine* baru, *middleware* berperan untuk memilih server terbaik dan menerjemahkan permintaan pengguna dalam bentuk permintaan yang dimengerti oleh *hypervisor* yang terinstal pada *server* terbaik. Penjelasan secara umum arsitektur sistem akan diuraikan pada Gambar 3.2.



Gambar 3.2: Desain Umum Sistem

3.2.2 Desain *Middleware*

Middleware digunakan sebagai penerjemaah permintaan dari pengguna agar dikenali oleh *hypervisor*. Selain itu *Middleware* juga bertugas untuk memilih *server* terbaik untuk alokasi *virtual machine*. *Middleware* terdari berbagi layanan yaitu, *HTTP Rest API*, basis data, *taks queue* dan *worker* yang bertugas menjalankan pekerjaan yang disimpan di *queue*. Arsitektur *Middleware* dapat dilihat pada Gambar 3.3.



Gambar 3.3: Desain *Middleware*

HTTP Rest API dibangun menggunakan Flask yang sudah terintegrasi dengan modul Pyvmomi dan Proxmoxer. Modul Pyvmomi dan Proxmoxer digunakan sebagai penghubung antara *Middleware* dengan *hypervisor*. Basis data MySQL terhubung dengan *HTTP Rest APi* sebagai tempat penyimpanan data dari sistem, seperti data pengguna, data *host* yang sudah terinstal *hypervisor* dan sebagainya. Untuk mengeksekusi permintaan dari pengguna, *HTTP Rest API* terhubung Celery sebagai *task queue* dan sebagai *worker*. Celery terhubung dengan basis data Redis sebagai tempat penyimpanan *queue*. Lalu yang terakhir terakhir terdapat *script* Python yang digunakan untuk menentukan *server* terbaik menggunakan algoritma *Analytic Hierarchy Process (AHP)*.

3.2.3 Perancangan *Task Queue*

Pada sistem ini, akan banyak proses yang berjalan dalam jangka waktu yang panjang karena banyak eksekusi perintah didalamnya. Jika proses tersebut dipanggil melalui protokol HTTP, maka umpan balik yang diberikan menunggu semua proses yang berkaitan dengan penggunaan dari pengguna selesai semua. Hal tersebut membuat pengguna yang melakukan permintaan perlu menunggu sampai selesai dan hal tersebut tidak efisien. Untuk mengatasi hal tersebut, proses yang memerlukan banyak perintah, akan dimasukkan ke dalam sebuah *queue* atau yang bisa disebut sebagai *task queue*. Untuk *task queue* menggunakan modul Celery yang menggunakan basis data Redis sebagai wadah untuk menampung perintah atau fungsi yang akan dikerjakan dalam bentuk *queue*. Modul Celery dijalankan sebagai layanan tersendiri dan terintegrasikan dengan Flask. Pada modul Celery secara *default* menyediakan 4 *worker* yang digunakan untuk mengerjakan perintah atau fungsi yang ditampung pada *queue*. *Worker* tersebut akan menjalankan perintah secara bergantian sampai tidak ada lagi perintah atau fungsi yang berada di *queue*.

3.2.4 Perancangan Alokasi *Virtual Machine Baru*

Alokasi *virtual machine* merupakan salah satu bagian dari fungsi dari sistem yang dibuat oleh penulis. Pengalokasian *virtual machine* baru pada *host atau server* yang tersedia melalui proses perhitungan algoritma AHP untuk menentukan *host atau server* terbaik. Setelah mendapatkan *server* terbaik, sistem akan mengambil informasi *server* terbaik pada basis data untuk mengetahui *hypervisor* yang terinstal. Untuk pengalokasian *virtual machine* baru pada Proxmox, sistem mengirimkan perintah melalui protokol SSH dan melalui modul Proxmoxer. Sedangkan untuk pengalokasian pada Vmware Vsphere, sistem

mengirimkan perintah melalui modul Pyvmomi. Untuk melakukan pengaturan IP pada *virtual machine*, sistem mengirimkan perintah melalui protokol SSH ke *virtual machine* yang baru saja dibuat terkecuali untuk virtual machine yang terinstal sistem operasi Ubuntu pada *hypervisor* Vmware Vsphere dengan menggunakan Pyvmomi.

3.2.5 Desain Web *Interface*

Pada sistem ini, pengguna dapat mengakses melalui dua *interface* yaitu web dan *command line interface*. *Interface* web pada sistem dibuat menggunakan kerangka kerja Laravel yang menggunakan bahasa pemrogramaan PHP. Web dapat diakses oleh *end-user* dan pengembang. Pada sistem ini web menjadi *interface* yang paling penting karena pengguna dapat melakukan pengaturan terhadap sistem seperti memanajemen host, memanajemen user, memanejemen *template*, memanajemen versi sistem operasi yang didukung, memanajemen kategori *resource* yang digunakan untuk menentukan *resource virtual machine*, memanajemen *virtual machine*, melihat riwayat pekerjaan dan memanajemen *API Secret Key*. Setiap permintaan pengguna melalui *interface* web akan dihubungkan langsung dengan *HTTP Rest API* sesuai *parameter* permintaan yang sudah ditentukan oleh *HTTP Rest API*. Umban Balik dari setiap permintaan pengguna, akan ditampilkan oleh web termasuk apabila *paramter* permintaan.

3.2.6 Desain *Command Line Interface*

Selain *interface* web, sistem ini dapat diakses melalui *Command Line Interface*. *Command Line Interface* dapat diinstal pada komputer pengguna. *Command Line Interface* dibuat menggunakan bahasa pemrograman Python. *End-user* dan pengembang hanya dapat melakukan berbagai pekerjaan saja,

seperti melihat *virtual machine*, mematikan atau menyalakan *virtual machine*, menghapus *virtual machine*, melihat kategori *resource* dan melihat sistem operasi yang tersedia. Setiap permintaan melalui *Command Line Interface* akan diteruskan ke *HTTP Rest API* sesuai *paramter* permintaan yang dibutuhkan. Untuk *Command Line Interface* menggunakan menggunakan *API Secret Key* yang sudah dibuat melalui *interface* web. *API Secret Key* digunakan untuk autentifikasi dan otorisasi hak akses pada sistem.

(Halaman ini sengaja dikosongkan)

BAB IV

IMPLEMENTASI

Bab ini membahas implementasi sistem Pengendali Elastisitas secara rinci. Pembahasan dilakukan secara rinci untuk setiap komponen yang ada, yaitu: *middleware*, alokasi *virtual machine* baru, *interface* web dan *command line interface*.

4.1 Lingkungan Implementasi

Dalam mengimplementasikan sistem, digunakan beberapa perangkat pendukung sebagai ebrikut.

4.1.1 Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. *Middleware*, *processor* Intel(R) Core(TM) i5 CPU @ 2.5 GHz dan Ram 4GB.
2. *Server* Proxmox dengan IP 10.151.36.222, *processor* Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz dan Ram 8GB.
3. *Server* Proxmox dengan IP 10.151.38.100, *processor* Intel(R) Xeon(R) CPU E5507 @ 2.27GHz dan Ram 8GB.
4. *Server* Vmware ESXI dengan IP 10.199.14.150, *processor* Intel(R) Xeon(R) CPU E5420 @ 2.50GHz dan Ram 4GB.
5. *Server* Windows Server 2016 dan terinstal Vmware Vcenter dengan IP 10.151.38.38, *processor* Intel(R) Core(TM) i7 CPU @ 2.50GHz dan Ram 16GB.

4.1.2 Perangkat Keras

Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut:

- Sistem Operasi Mac OS High Sierra 10.13.4
- Proxmox

- Vmware EXSI
- Vmware Vcenter
- Windows Server 2016
- Python 2.7
- Redis
- MySQL
- Flask
- Celery
- Insomnia

4.2 Implementasi *Middleware*

Server Middleware merupakan *server* inti dari sistem. Pada *server* ini yang akan mengelola keseluruhan data dari sistem. Pada server ini, selain mengelola kesuluruhan data, *server middleware* bertindak untuk mengambil keputusan dalam alokasi *virtual machine* baru. Selain itu semua permintaan dari pengguna akan diterjemahkan dalam bentuk *parameter* yang dibutuhkan untuk meneruskan permintaan dari pengguna ke hypervisor. *Server middleware* memilik IP 10.151.36.4 dan dapat diakses menggunakan port 5000.

4.2.1 Skema Basis Data Middleware Menggunakan MySQL

Untuk mengelola data yang ada pada sistem, dibutuhkan basis data sebagai tempat penyimpanannya, yaitu MySQL. MySQL *server* yang digunakan adalah versi 5.7.22. Data yang disimpan pada basis data adalah data *hypervisor*, data *host* yang sudah terinstal *hypervisor*, data sistem operasi yang didukung oleh sistem, data versi sistem operasi, data riwayat pekerjaan, data kategori *resource*, data *template* sistem operasi, data *users*, data *virtual machine*, data relasi kepemilikan *virtual machine* dengan pengguna dan data *API Secret Key*.

4.2.1.1 Tabel *Hypervisors*

Pada tabel *hypervisors* menyimpan data-data *hypervisor* yang didukung oleh sistem. Nama *hypervisor* tidak di-*hardcode* pada sistem agar dapat dilakukan pengembangan kedepannya. Berikut definisi tabel *hypervisors* pada Tabel 4.1.

Tabel 4.1: Tabel Hypervisors

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*name*	varchar(45)	Menunjukan nama *hypervisor* yang didukung oleh sistem.

4.2.1.2 Tabel *OS Distributions*

Pada tabel *os_distributions* menyimpan data-data distribusi sistem operasi yang didukung oleh sistem. Nama distribusi sistem operasi tidak di-*hardcode* pada sistem agar dapat dilakukan pengembangan kedepannya. Berikut definisi tabel *os_distributions* pada Tabel 4.2.

Tabel 4.2: Tabel OS Distributions

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.

Tabel 4.2: Tabel OS Distributions

No	Kolom	Tipe	Keterangan
2	*name*	varchar(45)	Menunjukan nama distribusi sistem operasi yang didukung oleh sistem.
3	logo	longtext	Menunjukan logo distribusi sistem operasi yang didukung oleh sistem. Disimpan dalam bentuk gambar yang sudah di-*endcode base64*.

4.2.1.3 Tabel *Request Categories*

Pada tabel *request_categories* menyimpan data-data kategori *resource* yang digunakan untuk menentukan *resource virtual machine*. Berikut definisi tabel *request_categories* pada Tabel 4.3.

Tabel 4.3: Tabel *Request Categories*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*name*	varchar(45)	Menunjukan nama kategori *resource*.

Tabel 4.3: Tabel *Request Categories*

No	Kolom	Tipe	Keterangan
3	*storage*	varchar(45)	Menunjukan besarnya *storage* yang digunakan untuk pembuatan *virtual machine*.
4	*memory*	varchar(45)	Menunjukan besarnya *memory* yang digunakan untuk pembuatan *virtual machine*.
5	*core*	varchar(45)	Menunjukan besarnya *core cpu* yang digunakan untuk pembuatan *virtual machine*.

4.2.1.4 Tabel *Users*

Pada tabel *users* menyimpan data-data pengguna yang dapat mengakses sistem. Berikut definisi tabel *users* pada Tabel 4.4.

Tabel 4.4: Tabel *Users*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*username*	varchar(45)	Menunjukan *username* pengguna yang digunakan untuk *login*.

Tabel 4.4: Tabel *Users*

No	Kolom	Tipe	Keterangan
3	*password*	varchar(100)	Menunjukan *password* pengguna yang digunakan untuk *login*.
4	*name*	varchar(45)	Menunjukan nama dari pengguna.
5	*is_admin*	int	Menunjukan jabatan dari pengguna. Apabila kolom *is_admin* tidak diisi maka akan bernilai 0.

4.2.1.5 Tabel *Hosts*

Pada tabel *hosts* menyimpan data-data *server* yang dapat digunakan oleh sistem untuk pembuatan *virtual machine*. Berikut definisi tabel *hosts* pada Tabel 4.5.

Tabel 4.5: Tabel *Hosts*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*ip*	varchar(45)	Menunjukan ip *server*.
3	*username*	varchar(100)	Menunjukan *username* yang digunakan untuk masuk ke server.
4	*password*	varchar(200)	Menunjukan *password* yang digunakan untuk masuk ke server.

Tabel 4.5: Tabel *Hosts*

No	Kolom	Tipe	Keterangan
5	*hypervi-sors_id*	char(36)	Menunjukan *foreign key* dari tabel *hypevisors*.
6	*node_name*	varchar(45)	Menunjukan nama dari server pada *hypervisor*.
7	*is_active*	int	Menunjukan status aktif atau tidaknya sebuah *server*. Apabila kolom *is_active* tidak diisi maka akan bernilai 1.

4.2.1.6 Tabel OS

Pada tabel *os* menyimpan data-data versi sistem operasi yang didukung oleh sistem. Berikut definisi tabel *os* pada Tabel 4.6.

Tabel 4.6: Tabel OS

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*name*	varchar(45)	Menunjukan versi sistem operasi.
3	*kategori*	int	Menunjukan kategori sistem operasi. Apabila kolom *kategori* tidak diisi maka akan bernilai 1 yang berarti sistem operasi Linux.

Tabel 4.6: Tabel OS

No	Kolom	Tipe	Keterangan
4	*os_distri-butions_id*	char(36)	Menunjukan *foreign key* dari tabel *os_distributions*.

4.2.1.7 Tabel *Templates*

Pada tabel *templates* menyimpan data-data *template* sistem operasi yang didukung oleh sistem. Berikut definisi tabel *templates* pada Tabel 4.7.

Tabel 4.7: Tabel *Templates*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*file*	varchar(45)	Menunjukan nama *template*.
3	*hypervi-sors_id*	char(36)	Menunjukan *foreign key* tabel *hypervisors*.
4	*os_id*	char(36)	Menunjukan *foreign key* tabel os.
5	*username*	varchar(45)	Menunjukan *username* sistem operasi pada template dengan hak akses *superuser template*.
5	*password*	varchar(100)	Menunjukan *username* sistem operasi pada template dengan hak akses *superuser template*.

Tabel 4.7: Tabel *Templates*

No	Kolom	Tipe	Keterangan
7	*username_default_user*	varchar(45)	Menunjukan *username* sistem operasi pada template untuk pengguna sistem.
8	*password_default_user*	varchar(45)	Menunjukan *password* sistem operasi pada template untuk pengguna sistem.

4.2.1.8 Tabel VMS

Pada tabel *vms* menyimpan data-data *virtual machine* yang terdaftar di sistem. Berikut definisi tabel *vms* pada Tabel 4.8.

Tabel 4.8: Tabel *VMS*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*memory*	varchar(45)	Menunjukan *memory* yang diatur pada *virtual machine*.
3	*storage*	varchar(45)	Menunjukan *storage* yang diatur pada *virtual machine*.
4	*core*	varchar(45)	Menunjukan *core* yang diatur pada *virtual machine*.

Tabel 4.8: Tabel *VMS*

No	Kolom	Tipe	Keterangan
5	*ip*	varchar(45)	Menunjukan ip yang diatur pada *virtual machine*.
6	*mac address*	varchar(45)	Menunjukan *mac address virtual machine*.
7	*hypervisors_id*	char(36)	Menunjukan *foreign key* tabel *hypervisors*.
8	*unique_id*	varchar(255)	Menunjukan nama unik dari *virtual machine*.
9	*name*	varchar(45)	Menunjukan nama *virtual machine*.
10	*host_id*	char(36)	Menunjukan *foreign key* tabel *hosts*.
11	*created_at*	timestamp	Menunjukan waktu pembuatan *virtual machine*.
12	*request_categories_id*	char(36)	Menunjukan *foreign key* tabel *request_categories*.
13	*os_id*	char(36)	Menunjukan *foreign key* tabel os.
14	*bridge_hypervisor*	varchar(45)	Menunjukan nama *interface bridge* pada *hypervisor*.
15	*nid_hypervisor*	varchar(45)	Menunjukan nama *network identifier* pada *hypervisor*.

Tabel 4.8: Tabel *VMS*

No	Kolom	Tipe	Keterangan
16	*random*	int	Menunjukan angka acak yang digunakan untuk identitas unik *virtual machine*.

4.2.1.9 Tabel *VM Owners*

Pada tabel *vm_owners* menyimpan data-data relasi antara *virtual machine* dengan pengguna yang terdaftar di sistem. Berikut definisi tabel *vms* pada Tabel 4.9.

Tabel 4.9: Tabel *VMS*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*vms_id*	char(36)	Menunjukan *foreign key* tabel *hypervisors*.
3	*users_id*	char(36)	Menunjukan *foreign key* tabel *users*.
4	*is_real_owner*	int	Menunjukan status kepemilikan *virtual machine*. Apabila tidak diisi memiliki nilai 0.

4.2.1.10 Tabel *Tasks*

Pada tabel *tasks* menyimpan data-data riwayat pekerjaan yang pernah atau sedang berjalan pada sistem. Berikut definisi tabel *tasks* pada Tabel 4.10.

Tabel 4.10: Tabel *Tasks*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*vms_id*	char(36)	Menunjukan *foreign key* tabel *hypervisors*.
3	*users_id*	char(36)	Menunjukan *foreign key* tabel *users*.
4	*status*	int	Menunjukan status pekerjaan. Status bernilai 0 apabila pekerjaan masih berjalan, bernilai 1 apabila perkerjaan selesai dan sukses dan bernilai -1 apabila pekerjaan gagal dikerjakan.
5	*start*	timestamp	Menunjukan waktu pekerjaan dimulai.
6	*end*	timestamp	Menunjukan waktu pekerjaan berakhir.
7	*description*	text	Menunjukan status pesan pekerjaan.
8	*celery_id*	varchar(45)	Menunjukan id pekerjaan pada *celery queue*.

Tabel 4.10: Tabel *Tasks*

No	Kolom	Tipe	Keterangan
9	*last_step*	int	Menunjukan langkah terakhir yang dikerjaan oleh sistem.
10	*action*	varchar(255)	Menunjukan nama pekerjaan yang dilakukan.

4.2.1.11 Tabel *Tokens*

Pada tabel *tokens* menyimpan data-data *API Secret Key*. Berikut definisi tabel *tokens* pada Tabel 4.11.

Tabel 4.11: Tabel *Tasks*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*users_id*	char(36)	Menunjukan *foreign key* tabel *users*.
3	*token*	longtext	Menunjukan *API Secret Key*.
4	*name*	timestamp	Menunjukan nama *API Secret Key*.
5	*is_write*	int	Menunjukan hak akses *API Secret Key*. Ketika kolom *is_write* bernilai 1 (satu), maka *API Secret Key* dapat melakukan aksi pengubahan data.

4.2.1.12 Tabel IP *Addresses*

Pada tabel *ip_addresses* menyimpan data-data *pool* ip yang dapat digunakan pada suatu *server*. Berikut definisi tabel *ip_address* pada Tabel 4.12.

Tabel 4.12: Tabel *IP Addresses*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*hosts_id*	char(36)	Menunjukan *foreign key* tabel *hosts*.
3	*ip*	varchar(45)	Menunjukan *API Secret Key*.
4	*netmask*	varchar(45)	Menunjukkan *netmask* suatu ip.
5	*gateway*	varchar(45)	Menunjukan *gateway* suatu ip.

4.2.2 Implementasi Autentifikasi dan Otorisasi pada *Middleware*

Untuk mengakses *middleware* diperlukan *token* yang dikirimkan pada *header* paket. *Token* didapatkan setelah *login* pada *middleware*. Terdapat dua mekanisme autentifikasi dan otorisasi yaitu dengan mengirimkan data *username* dan *password* atau menggunakan *API Secret Key* yang didapatkan melalui *interface* web.

4.2.2.1 Implementasi Autentifikasi dan Otorisasi dengan *username* dan *password*

Untuk melakukan autentifikasi dan otorisasi menggunakan *username* dan *password*, pengguna dapat mengakses pada *end-point* `/api/v1/auth/login/form` dengan menggunakan *Method Post*. *Parameter* yang dibutuhkan untuk mengakses *end-point* tersebut adalah *username* dan *password*.

4.2.2.2 Implementasi Autentifikasi dan Otorisasi dengan *API Secret Key*

Untuk melakukan autentifikasi dan otorisasi menggunakan *API Secret Key*, pengguna dapat mengakses pada *end-point* `/api/v1/auth/login/token` dengan menggunakan *Method Post*. *Parameter* yang dibutuhkan untuk mengakses *end-point* tersebut adalah *token*.

4.2.2.3 Implementasi *Token* untuk Autentifikasi dan Otorisasi

Token dikirim oleh *middleware* dalam bentuk *response* dari permintaan autentifikasi dan otorisasi oleh pengguna yang berupa *JSON*. Bentuk *response* dapat dilihat pada Gambar 4.1.


```
{
	"data":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6IjUxMTQxMDAwMjAiLCJ1c2VyX3Rva2VuIjpu
dWxsLCJ1YSI6Imluc29tbmlhLzUuMTYuMiIsImlzX2FkbWluIjoxLCJleHAiOjE1MjcyNjU4NzUsImlzX3ZpYV90b2
tlbiI6MCwiaXAiOiIxMjcuMC4wLjEiLCJpc192aWFfd2ViIjoxLCJpZCI6Ijg2YjRlNDYwLTE2MmEtMTFlOC1hNjRl
LWE4N2M3M2NjYTZkOSJ9.eqbi726YrgL82FTynf74JY3Tk8UiyzubBa6SYrJFwPk",
	"status": 200
}
```


Gambar 4.1: *Response Token* dari *Middleware*

Token dikirimkan oleh *middleware* pada *index JSON* dengan nama data dan dalam bentuk *JSON Web Token (JWT)*. Ketika *token* tersebut di-*decode*, maka *token* berisi detail pengguna yang sudah terautentifikasi dan terotorisasi.

4.2.3 Implementasi *Endpoint* pada *Middleware*

Untuk mempermudah manajemen sistem, penulis menyediakan *end-point* pada *middleware*.

4.2.3.1 *End-Point* Manajemen *Host*

Untuk manajemen *host*, terdapat lima operasi yang didapat dilakukan yaitu melihat daftar *host*, menambah *host*, melihat detail data *host*, mengubah data *host* dan menghapus *host*. Untuk detail *end-point* dapat dilihat pada Tabel 4.13.

Tabel 4.13: Tabel *End-point* Manajemen *Host*

No	*End-point*	*Method*	*Keterangan*
1	`/api/v1/hypervisor/<hypervisor_id>/host`	*GET*	Untuk melihat daftar *host* pada *hypervisor*.
2	`/api/v1/hypervisor/<hypervisor_id>/host/create`	POST	Untuk menambah *host*.
3	`/api/v1/hypervisor/<hypervisor_id>/host/<host_id>`	GET	Untuk melihat data *host*.

Tabel 4.13: Tabel *End-point* Manajemen *Host*

No	*End-point*	*Method*	*Keterangan*
4	`/api/v1/hypervisor/<hypervisor_id>/host/<host_id>`	PUT	Untuk mengubah data *host*.
5	`/api/v1/hypervisor/<hypervisor_id>/host/<host_id>`	DELETE	Untuk menghapus data *host*.

4.2.3.2 *End-Point* Manajemen Kategori *Resource*

Untuk manajemen kategori *resource*, terdapat lima operasi yang didapat dilakukan yaitu melihat daftar kategori *resource*, menambah kategori *resource*, melihat detail data kategori *resource*, mengubah data kategori *resource* dan menghapus kategori *resource*. Untuk detail *end-point* dapat dilihat pada Tabel 4.14.

Tabel 4.14: Tabel *End-point* Manajemen Kategori *Resource*

No	*End-point*	*Method*	*Keterangan*
1	`/api/v1/requestcategory`	*GET*	Untuk melihat daftar kategori *resource*.
2	`/api/v1/requestcategory/create`	POST	Untuk menambah kategori *resource*.
3	`/api/v1/requestcategory/<resource_id>`	GET	Untuk melihat data kategori *resource*.

Tabel 4.14: Tabel *End-point* Manajemen Kategori *Resource*

No	*End-point*	*Method*	*Keterangan*
4	`/api/v1/requestcategory/<resource_id>`	PUT	Untuk mengubah data kategori *resource*.
5	`/api/v1/requestcategory/<resource_id>`	DELETE	Untuk menghapus data kategori *resource*.

4.2.3.3 *End-Point* Manajemen Versi Sistem Operasi

Untuk manajemen versi sistem operasi, terdapat lima operasi yang didapat dilakukan yaitu melihat daftar versi sistem operasi, menambah versi sistem operasi, melihat detail data sistem operasi, mengubah data versi sistem operasi dan menghapus versi sistem operasi. Untuk detail *end-point* dapat dilihat pada Tabel 4.15.

Tabel 4.15: Tabel *End-point* Manajemen Versi Sistem Operasi

No	*End-point*	*Method*	*Keterangan*
1	`/api/v1/os`	*GET*	Untuk melihat daftar versi sistem operasi.
2	`/api/v1/os/create`	POST	Untuk menambah versi sistem operasi.
3	`/api/v1/os/<os_id>`	GET	Untuk melihat data versi sistem operasi.
4	`/api/v1/os/<os_id>`	PUT	Untuk mengubah data versi sistem operasi.
5	`/api/v1/os/<os_id>`	DELETE	Untuk menghapus data suatu versi sistem operasi.

4.2.3.4 *End-Point* Manajemen *Template* Sistem Operasi

Untuk manajemen *template* sistem operasi, terdapat lima operasi yang didapat dilakukan yaitu melihat daftar *template* sistem operasi, menambah *template* sistem operasi, melihat detail data *template* sistem operasi, mengubah data *template* sistem operasi dan menghapus *template* sistem operasi. Untuk detail *end-point* dapat dilihat pada Tabel 4.16.

Tabel 4.16: Tabel *End-point* Manajemen *Template* Sistem Operasi

No	*End-point*	*Method*	*Keterangan*
1	`/api/v1/hypervisor/<hypervisor_id>/template`	*GET*	Untuk melihat daftar *template* sistem operasi.
2	`/api/v1/hypervisor/<hypervisor_id>/template/create`	POST	Untuk menambah *template* sistem operasi.
3	`/api/v1/hypervisor/<hypervisor_id>/template/<template_id>`	GET	Untuk melihat data *template* sistem operasi.
4	`/api/v1/hypervisor/<hypervisor_id>/template/<template_id>`	PUT	Untuk mengubah data *template* sistem operasi.

Tabel 4.16: Tabel *End-point* Manajemen *Template* Sistem Operasi

No	*End-point*	*Method*	*Keterangan*
5	`/api/v1/hypervisor/<hypervisor_id>/template/<template_id>`	DELETE	Untuk menghapus data *template* sistem operasi.

4.2.3.5 *End-Point* Manajemen *User*

Untuk manajemen pengguna, terdapat lima operasi yang didapat dilakukan yaitu melihat daftar pengguna, menambah pengguna, melihat detail data pengguna, mengubah data pengguna dan menghapus pengguna. Untuk detail *end-point* dapat dilihat pada Tabel 4.17.

Tabel 4.17: Tabel *End-point* Manajemen *User*

No	*End-point*	*Method*	*Keterangan*
1	`/api/v1/user`	*GET*	Untuk melihat daftar pengguna.
2	`/api/v1/user/create`	POST	Untuk menambah pengguna.
3	`/api/v1/user/<user_id>`	GET	Untuk melihat pengguna.
4	`/api/v1/user/<user_id>`	PUT	Untuk mengubah pengguna.
5	`/api/v1/user/<user_id>`	DELETE	Untuk menghapus pengguna.

4.2.3.6 *End-Point* Manajemen *API Secret Key*

Untuk manajemen *API secret key*, terdapat lima operasi yang didapat dilakukan yaitu melihat daftar *API secret key*, menambah *API secret key*, melihat detail data *API secret key*, mengubah data *API secret key* dan menghapus *API secret key*. Untuk detail *end-point* dapat dilihat pada Tabel 4.20.

Tabel 4.18: Tabel *End-point* Manajemen *API Secret Key*

No	*End-point*	*Method*	*Keterangan*
1	`/api/v1/user/<user_id>/token`	*GET*	Untuk melihat daftar *API secret key*.
2	`/api/v1/user/<user_id>/token/create`	POST	Untuk menambah *API secret key*.
3	`/api/v1/user/<user_id>/token/<token_id>`	GET	Untuk melihat *API secret key*.
4	`/api/v1/user/<user_id>/token/<token_id>`	PUT	Untuk mengubah *API secret key*.
5	`/api/v1/user/<user_id>/token/<token_id>`	DELETE	Untuk menghapus *API secret key*.

4.2.3.7 *End-Point* Manajemen *Virtual Machine*

Untuk manajemen *API secret key*, terdapat 12 operasi yang didapat dilakukan yaitu melihat daftar *API secret key*, menambah

virtual machine, melihat detail *virtual machine*, meningkatkan kategori *resource virtual machine*, menghapus *virtual machine*, menyalakan *virtual machine*, mematikan *virtual machine*, melihat pengguna *virtual machine*, membagikan *virtual machine* ke pengguna lain, menghapus pengguna dari hak kepemilikan *virtual machine*, melihat status *virtual machine* dan melihat riwayat *virtual machine*. Untuk detail *end-point* dapat dilihat pada Tabel 4.21.

Tabel 4.19: Tabel *End-point* Manajemen *Virtual Machine*

No	***End-point***	***Method***	***Keterangan***
1	`/api/v1/vm`	*GET*	Untuk melihat daftar *virtual machine*.
2	`/api/v1/vm/create`	POST	Untuk menambah *virtual machine*.
3	`/api/v1/vm/<vm_id>/status/start`	POST	Untuk menyalakan *virtual machine*.
4	`/api/v1/vm/<vm_id>/status/stop`	POST	Untuk mematikan *virtual machine*.
5	`/api/v1/vm/<vm_id>/owner`	GET	Untuk melihat daftar kepemilikan *virtual machine*.
6	`/api/v1/vm/<vm_id>`	DELETE	Untuk menghapus *virtual machine*.
7	`/api/v1/vm/<vm_id>/owner`	DELETE	Untuk menghapus pengguna dari kepemilikan *virtual machine*.

Tabel 4.19: Tabel *End-point* Manajemen *Virtual Machine*

No	*End-point*	*Method*	*Keterangan*
8	`/api/v1/vm/<vm_id>/owner`	POST	Untuk membagikan hak kepemilikan *virtual machine* ke pengguna lain.
9	`/api/v1/vm/<vm_id>/status/current`	GET	Untuk melihat status *virtual machine*.
10	`/api/v1/vm/<vm_id>`	GET	Untuk melihat detail *virtual machine*.
11	`/api/v1/vm/<vm_id>`	PUT	Untuk me-*resize* kategori *resource virtual machine*.
12	`/api/v1/vm/<vm_id>/history`	GET	Untuk melihat riwayat *virtual machine*.

4.2.3.8 *End-Point* Manajemen *API Secret Key*

Untuk manajemen *API secret key*, terdapat lima operasi yang didapat dilakukan yaitu melihat daftar *API secret key*, menambah *API secret key*, melihat detail data *API secret key*, mengubah data *API secret key* dan menghapus *API secret key*. Untuk detail *end-point* dapat dilihat pada Tabel 4.20.

Tabel 4.20: Tabel *End-point* Manajemen *API Secret Key*

No	*End-point*	*Method*	*Keterangan*
1	`/api/v1/user/<user_id>/token`	*GET*	Untuk melihat daftar *API secret key*.

Tabel 4.20: Tabel *End-point* Manajemen *API Secret Key*

No	***End-point***	***Method***	***Keterangan***
2	`/api/v1/user/<user_id>/token/create`	POST	Untuk menambah *API secret key*.
3	`/api/v1/user/<user_id>/token/<token_id>`	GET	Untuk melihat *API secret key*.
4	`/api/v1/user/<user_id>/token/<token_id>`	PUT	Untuk mengubah *API secret key*.
5	`/api/v1/user/<user_id>/token/<token_id>`	DELETE	Untuk menghapus *API secret key*.

4.2.3.9 *End-Point* Melihat *Task*

Untuk melihat daftar *task*, pengguna dapat mengakses *end-point /api/v1/task* dengan *method GET*.

4.2.4 Implementasi Integrasi *HTTP Rest API* dengan Celery *Task Queque*

Pada sistem ini menggunakan *Task Queue* untuk mengoptimalkan performa. *Task Queue* dibangun menggunakan modul Python Celery yang terintegrasi dengan *HTTP Rest API* yang dibangun menggunakan Python Flask. Untuk melakukan instalasi Python Celery dapat dilihat pada Kode Sumber IV.1.

```
pip install celery
```

Kode Sumber IV.1: Perintah Instalasi Python Celery

Setelah melakukan instalasi Python Celery, langkah selanjutnya mengintegrasikan *object* Python Flask dengan Python Celery sebagai konfigurasi dasar. *Pseudocde* Untuk pengintegrasian dapat dilihat pada Kode Sumber IV.2.

```
from celery import Celery
FUNCTION make_celery(app)
  celery <- Celery(app.import_name,
    backend=app.config['result_backend'],
    broker=app.config['CELERY_BROKER_URL'])
  celery.conf.update(app.config)
  celery.conf.task_default_queue <- 'hypgen_queue
     '
  TaskBase <- celery.Task
  CLASS ContextTask(TaskBase):
    abstract <- True
    FUNCTION __call__(self, *args, **kwargs)
      with app.app_context()
        RETURN TaskBase__call__(self, *args, **
           kwargs)
    ENDFUNCTION
  ENDCLASS
  celery.Task <- ContextTask
  RETURN celer
ENDFUNCTION
```

Kode Sumber IV.2: *Pseudocode* Pengintegrasian Python Flask dan Python Celery

Setelah melakukan pengintegrasian, selanjutnya membuat *file Tasks.py* yang berisi daftar pekerjaan yang dikerjakan

menggunakan Python Celery. *Pseudocode file Tasks.py* dapat dilihat pada Kode Sumber IV.3.

```
from system.App import flask_app as app
from system.Celery_App import make_celery

celery_worker <- make_celery(app)

@celery_worker.task(bind <- True)
FUNCTION create_vm(self, params <- {}):
  return status, message, params
ENDFUNCTION
```

Kode Sumber IV.3: *Pseudocode File Tasks.py*

Variabel *celery_worker* merupakan variabel hasil pengintegrasian Python Flask dengan Python Celery. Variabel tersebut dijadikan fungsi *decorator* agar fungsi dibawahnya dapat dikerjakan dengan Python Celery. Untuk cara pemanggilan fungsi dapat dilihat pada Kode Sumber IV.4.

```
from app.Library import Tasks
FUNCTION create_vm()
  params <- getUserParams()
  Tasks.create_vm.delay(params)
ENDFUNCTION
```

Kode Sumber IV.4: Pseudocode Membuat *Virtual Machine* pada *File VM_Controller.py*

Untuk memanggil fungsi pada file *Tasks.py*, pada *controller* ditambahkan *method delay* agar saat pemanggilan fungsi tersebut dimasukkan ke dalam *Task Queue* dan *task* bersifat asinkronus. Python Celery berjalan diatas proses tersendiri. Untuk menjalankan Python Celery dengan menjalankan perintah yang

dapat dilihat pada Kode Sumber IV.5.

```
celery worker –E ––app=app.Library.Tasks.
    celery_worker –Q hypgen_queue ––loglevel
    =DEBUG
```

Kode Sumber IV.5: Perintah Untuk Menjalankan Python Celery

4.2.5 Implementasi Manajemen *Virtual Machine*

Melalui *middleware* pengguna dapat membuat *virtual machine* baru, menyalakan *virtual machine*, mematikan *virtual machine*, meng-*resize resource virtual machine* dan menghapus *virtual machine* tanpa berinteraksi secara langsung dengan *hypervisor*. *Middleware* akan menerjemahkan permintaan pengguna dalam bentuk *parameter-parameter* yang dibutuhkan untuk terhubung ke *hypervisor* secara otomatis.

4.2.5.1 Implementasi Alokasi *Virtual Machine* Baru

Untuk membuat *virtual machine*, pengguna mengirimkan *parameter* berupa nama *virtual machine*, jenis sistem operasi dan kategori *resource* melalui *end-point HTTP Rest API* yang sudah disediakan. *HTTP Rest API* akan menyimpan data *virtual machine* baru pengguna ke dalam basis data. Selain menyimpan data *virtual machine* baru, *HTTP Rest API* akan membuat *task* baru. *Pseudocode* dapat dilihat pada Kode Sumber IV.6.

```
1 FUNCTION create():
2   vm <- VM()
3
4   TRY
5     vm.save()
```

```
  CATCH
    RETURN "Error 500"
  ENDTRY

  vm_owner <- VM_Owner()

  TRY
    vm_owner.save()
  CATCH
    RETURN "Error 500"
  ENDTRY

  task <- TaskModel()

  TRY
    task.save()
  CATCH
    RETURN "Error 500"
  ENDTRY

  params <- {"name":
      getNameVirtualMachineFromUser(),"last_step":
       1 , "os":getOsFromUser(), "request_category
      ":getResourceFromUser(), "vm": vm, "owner_vm
      ": getUserUniqueID(), "random": timestamp, "
      task": task}

  Tasks.create_vm.delay(params)
  RETURN {"status":200, "data": "Proses pembuatan
       virtual machine baru akan dimasukkan dalam
      antrian. Mohon tunggu sampai proses selesai"
      }
ENDFUNCTION
```

Kode Sumber IV.6: *Pseudocode* Alokasi *Virtual Machine* Baru pada *File VM_Controller.py*

Setelah menyimpan data *virtual machine* dan membuat *task* baru, selanjutnya *middleware* akan memanggil fungsi *create_vm* pada *file Tasks.py* melalui fungsi *delay* agar *task* tersebut disimpan pada *queue* dan berjalan secara asinkronus. Pada fungsi *create_vm*, *middleware* akan memanggil fungsi *create_vm* pada *class Hypervisor_Library*. *Class Hypervisor_Library* bertugas untuk menghubungkan *middleware* dengan *hypervisor*.

Proses selanjutnya terjadi dalam fungsi *create_vm* pada *class Hypervisor_Library*, *middleware* akan melakukan *query* untuk mencari *server* yang tersedia berdasarkan sistem operasi. Sistem operasi memliki *template* disetiap *hypervisor* dan setiap *hypervisor* memiliki relasi ke *server*. Selanjutnya *middleware* akan melakukan *query* untuk mendapatkan data kategori *resource* untuk *resource virtual machine*. Setelah mendapatkan *server* yang tersedia dan data kategori *resource*, *middleware* memanggil fungsi *get_selected_host* untuk mendapatkan *server* terbaik dengan menggunakan algoritma AHP. Setelah mendapatkan data *server* terbaik, *middleware* akan menjalankan perintah *virtual machine* berdasarkan *hypervisor* dari *server* terbaik. Pseudocode fungsi *create_vm* pada *class Hypervisor_Library* dapat dilihat pada Kode Sumber IV.7.

```
1  FUNCTION create_vm(self, params <- {})
2
3    status, hosts <-  get_host_by_os(params["os"])
4
5    IF status = False
6    THEN
7      RETURN False, "error resource tidak ada",
         params
```

```
8   ENDIF
9
10  status, resource <-  get_data_request_category(
        params["request_category"])
11
12  IF status = False
13  THEN
14    RETURN False, "error tidak ketemu", params
15  ELSE:
16    params <- params + resource
17  ENDIF
18
19  status, message, params, selected_host <-
        get_selected_host(hosts, params)
20
21  IF status = False
22  THEN
23    RETURN status, message, params
24  ENDIF
25
26
27  IF selected_host.hypervisor = "proxmox"
28  THEN
29    status, message, params <- Proxmox_Library().
          create_vm(params, selected_host)
30  ELSE IF selected_host.hypervisor = "vmware"
31  THEN
32    status, message, params <- Vsphere_Library().
          create_vm(params, selected_host)
33  ENDIF
34
35  RETURN status, message, params
36 ENDFUNCTION
```

Kode Sumber IV.7: *Pseudocode* Fungsi *create_vm* pada pada *class Hypervisor_Library*

4.2.6 Implementasi Mematikan *Virtual Machine*

Untuk mematikan *virtual machine*, pengguna hanya perlu mengakses *end-point*. Pada *end-point* terdapat *path* yang akan diuraikan oleh *HTTP Rest API* sebagai *vm_id*. Selanjutnya *middleware* akan melakukan *query* untuk mendapatkan data-data mengenai *virtual machine* termasuk memvalidasi kepemilikan *virtual machine*.

Setelah mendapatkan data mengenai *virtual machine*, selanjutnya *middleware* akan membuat *task* baru. Kemudian data *virtual machine* dan data *task* dijadikan parameter pemanggilan fungsi *stop_vm* pada *file Tasks.py* oleh *middleware* dengan menambah fungsi *delay* agar *task* tersebut disimpan pada *queue* dan berjalan secara asinkronus. Pada fungsi *stop_vm*, *middleware* akan memanggil fungsi *stop_vm* pada *class Hypervisor_Library*. Pseudocode dapat dilihat pada Kode Sumber IV.8.

```
1  FUNCTION stop(id)
2
3    vm <- VM.select().where(VM.id = id).first()
4
5    IF vm = None
6    THEN
7      RETURN "Error 404"
8    ENDIF
9
10   status, message, token <- Token_Parser().parse
       ()
```

```
11
12    IF status  = False
13    THEN
14      RETURN "Error 400"
15    ENDIF
16
17    owner_status <- check_owner(id, token)
18
19    IIF owner_status  = False THEN
20      RETURN "Error 403"
21    ENDIF
22
23    task <- TaskModel()
24
25    TRY
26      task.save()
27    CATCH
28      RETURN "Error 500"
29    ENDTRY
30
31    params <- {'task': task, 'vm': vm}
32
33    Tasks.stop_vm.delay(params)
34
35    RETURN {"status":200, "data": "Proses mematikan
          akan dimasukkan dalam antrian. Mohon tunggu
          sampai proses selesai"}
36  ENDFUNCTION
```

Kode Sumber IV.8: *Pseudocode* Mematikan *Virtual Machine* pada *VM_Controller*

Proses selanjutnya terjadi dalam fungsi *stop_vm* pada *class Hypervisor_Library*, *middleware* akan memeriksa jenis *hypervisor* pada data *virtual machine* dengan *parameter* untuk

menentukan mekanisme mematikan *virtual machine*. Pseudocode fungsi *stop_vm* dapat dilihat pada Kode Sumber IV.9.

```
FUNCTION stop_vm(self, params <- {})
  vm = params['vm']
  IF vm.hypervisor = "proxmox"
  THEN
    status, message, params <- Proxmox_Library().
        stop_vm(params, params['vm']['hosts'])
  ELSE IF vm.hypervisor = "vmware"
  THEN
    status, message, params <- Vsphere_Library().
        stop_vm(params, vm.hosts)
  ENDIF

  RETURN status, message, params
ENDFUNCTION
```

Kode Sumber IV.9: *Pseudocode* Fungsi *stop_vm* pada pada *class Hypervisor_Library*

4.2.7 Implementasi Menyalakan *Virtual Machine*

Untuk menyalakan *virtual machine*, pengguna hanya perlu mengakses *end-point*. Pada *end-point* terdapat *path* yang akan diuraikan oleh *HTTP Rest API* sebagai *vm_id*. Selanjutnya *middleware* akan melakukan *query* untuk mendapatkan data-data mengenai *virtual machine* termasuk memvalidasi kepemilikan *virtual machine*.

Setelah mendapatkan data mengenai *virtual machine*, selanjutnya *middleware* akan membuat *task* baru. Kemudian data *virtual machine* dan data *task* dijadikan parameter pemanggilan fungsi *start_vm* pada *file Tasks.py* oleh *middleware* dengan

menambah fungsi *delay* agar *task* tersebut disimpan pada *queue* dan berjalan secara asinkronus. Pada fungsi *start_vm*, *middleware* akan memanggil fungsi *start_vm* pada *class Hypervisor_Library*. Pseudocode dapat dilihat pada Kode Sumber IV.10.

```
1  FUNCTION start(id)
2
3    vm <- VM.select().where(VM.id = id).first()
4
5    IF vm = None
6    THEN
7      RETURN "Error 404"
8    ENDIF
9
10   status, message, token <- Token_Parser().parse
         ()
11
12   IF status  = False
13   THEN
14     RETURN "Error 400"
15   ENDIF
16
17   owner_status <- check_owner(id, token)
18
19   IIF owner_status  = False THEN
20     RETURN "Error 403"
21   ENDIF
22
23   task <- TaskModel()
24
25   TRY
26     task.save()
```

```
27    CATCH
28      RETURN "Error 500"
29    ENDTRY
30
31    params <- {'task': task, 'vm': vm}
32
33    Tasks.start_vm.delay(params)
34
35    RETURN {"status":200, "data": "Proses
          menyalakan akan dimasukkan dalam antrian.
          Mohon tunggu sampai proses selesai"}
36  ENDFUNCTION
```

Kode Sumber IV.10: *Pseudocode* Menyalakan *Virtual Machine* pada *VM_Controller*

Proses selanjutnya terjadi dalam fungsi *start_vm* pada *class Hypervisor_Library*, *middleware* akan memeriksa jenis *hypervisor* pada data *virtual machine* dengan *parameter* untuk menentukan mekanisme menyalakan *virtual machine*. Pseudocode fungsi *start_vm* dapat dilihat pada Kode Sumber IV.11.

```
1   FUNCTION start_vm(self, params <- {})
2     vm = params['vm']
3
4     IF vm.hypervisor = "proxmox"
5     THEN
6       status, message, params <- Proxmox_Library().
            start_vm(params, params['vm']['hosts'])
7     ELSE IF vm.hypervisor = "vmware"
8     THEN
9       status, message, params <- Vsphere_Library().
            start_vm(params, vm.hosts)
```

```
10    ENDIF
11
12    RETURN status, message, params
13  ENDFUNCTION
```

Kode Sumber IV.11: *Pseudocode* Fungsi *start_vm* pada pada *class Hypervisor_Library*

4.2.8 Implementasi Menghapus *Virtual Machine*

Untuk menghapus *virtual machine*, pengguna hanya perlu mengakses *end-point*. Pada *end-point* terdapat *path* yang akan diuraikan oleh *HTTP Rest API* sebagai *vm_id*. Selanjutnya *middleware* akan melakukan *query* untuk mendapatkan data-data mengenai *virtual machine* termasuk memvalidasi kepemilikan *virtual machine*.

Setelah mendapatkan data mengenai *virtual machine*, selanjutnya *middleware* akan membuat *task* baru. Kemudian data *virtual machine* dan data *task* dijadikan parameter pemanggilan fungsi *delete_vm* pada *file Tasks.py* oleh *middleware* dengan menambah fungsi *delay* agar *task* tersebut disimpan pada *queue* dan berjalan secara asinkronus. Pada fungsi *delete_vm*, *middleware* akan memanggil fungsi *delete_vm* pada *class Hypervisor_Library*. Pseudocode dapat dilihat pada Kode Sumber IV.12.

```
1   FUNCTION delete_vm(id)
2
3     vm <- VM.select().where(VM.id = id).first()
4
5     IF vm = None
6     THEN
7       RETURN "Error 404"
8     ENDIF
```

```
9
10    status, message, token <- Token_Parser().parse
          ()
11
12    IF status  = False
13    THEN
14      RETURN "Error 400"
15    ENDIF
16
17    owner_status <- check_owner(id, token)
18
19    IF owner_status  = False THEN
20      RETURN "Error 403"
21    ENDIF
22
23    task <- TaskModel()
24
25    TRY
26      task.save()
27    CATCH
28      RETURN "Error 500"
29    ENDTRY
30
31    params <- {'task': task, 'vm': vm}
32
33    Tasks.start_vm.delay(params)
34
35  RETURN {"status":200, "data": "Proses
        pengahapusan virtual machine akan dimasukkan
        dalam antrian. Mohon tunggu sampai proses
        selesai"}
36  ENDFUNCTION
```

Kode Sumber IV.12: *Pseudocode* Menghapus *Virtual Machine* pada *VM_Controller*

Proses selanjutnya terjadi dalam fungsi *delete_vm* pada *class Hypervisor_Library*, *middleware* akan memeriksa jenis *hypervisor* pada data *virtual machine* dengan *parameter* untuk menentukan mekanisme menghapus *virtual machine*. Pseudocode fungsi *delete_vm* dapat dilihat pada Kode Sumber IV.13.

```
1  FUNCTION delete_vm(self, params <- {})
2      vm = params['vm']
3
4      IF vm.hypervisor = "proxmox"
5      THEN
6        status, message, params <- Proxmox_Library
             ().start_vm(params, params['vm']['hosts'
             ])
7      ELSE IF vm.hypervisor = "vmware"
8      THEN
9        status, message, params <- Vsphere_Library
             ().start_vm(params, vm.hosts)
10     ENDIF
11
12     RETURN status, message, params
13 ENDFUNCTION
```

Kode Sumber IV.13: *Pseudocode* Fungsi *delete_vm* pada pada *class Hypervisor_Library*

4.2.9 Implementasi *Resize Resource Virtual Machine*

Untuk melakukan *resize resource virtual machine*, pengguna mengakses *end-point* dengan mengirimkan parameter *request_category_id* sebagai kategori *resource* yang akan diatur pada *virtual machine*. Pada *end-point* terdapat *path* yang akan diuraikan oleh *HTTP Rest API* sebagai *vm_id*. Selanjutnya *middleware* akan melakukan *query* untuk mendapatkan data-data mengenai *virtual machine*, memvalidasi kepemilikan *virtual machine*, dan melakukan *query* untuk mendapatkan data kategori *resource* yang dikirim oleh pengguna.

Setelah mendapatkan data mengenai *virtual machine*, selanjutnya *middleware* akan membandingkan kategori *resource* yang dikirim oleh pengguna dengan kategori *resource* yang sudah diimplementasikan pada *virtual machine*. Apabila sama, *middleware* akan mengirim umpan balik berupa kode *HTTP* 400. Setelah itu *middleware* akan membuat *task* baru. Kemudian data *virtual machine* dan data *task* dijadikan parameter pemanggilan fungsi *resize_vm* pada *file Tasks.py* oleh *middleware* dengan menambah fungsi *delay* agar *task* tersebut disimpan pada *queue* dan berjalan secara asinkronus. Pada fungsi *resize_vm*, *middleware* akan memanggil fungsi *resize_vm* pada *class Hypervisor_Library*. Pseudocode dapat dilihat pada Kode Sumber IV.14.

```
1  FUNCTION vm_resize(id):
2    vm <- VM.select().where(VM.id = id).first()
3
4    IF vm = None
5    THEN
6      RETURN "Error 404"
7    ENDIF
8
```

```
9   IF vm.request_categories =
        getUserRequestParamter('request_category_id
        ')
10  THEN
11    RETURN 'Error 400, kategori resource sudah
          diimplementasikan'
12  ENDIF
13
14
15  status, message, token <- Token_Parser().parse
        ()
16
17  IF status = False
18  THEN
19    RETURN 'Error 400'
20  ENDIF
21
22  owner_status <- check_owner(id, token)
23
24  IF owner_status = False
25  THEN
26    RETURN 'Error 400'
27  ENDIF
28
29  request_category <- Request_Category.select().
        where(Request_Category.id =
        getUserRequestParamter('request_category_id
        ')).first()
30
31  IF request_category is None:
32    RETURN 'Error 404'
33  ENDIF
34
```

```
35    task <- TaskModel()
36
37
38    TRY
39      task.save()
40    CATCH Exception as e:
41      RETURN 'Error 500'
42    ENDTRY
43
44    params <- {"vm" : vm, "task" : task, "
          request_category": request_category}
45
46    Tasks.resize_vm.delay(params)
47
48    RETURN {"status":200, "data": "Proses resize vm
          akan dimasukkan dalam antrian. Mohon tunggu
          sampai proses selesai"}
49  ENDFUNCTION
```

Kode Sumber IV.14: *Pseudocode Resize Resource Virtual Machine* pada *VM_Controller*

Proses selanjutnya terjadi dalam fungsi *resize_vm* pada *class Hypervisor_Library*, *middleware* akan memeriksa jenis *hypervisor* pada data *virtual machine* dengan *parameter* untuk menentukan mekanisme *resize resource virtual machine*. Pseudocode fungsi *resize_vm* dapat dilihat pada Kode Sumber IV.15.

```
1  FUNCTION resize_vm(self, params <- {})
2  vm = params['vm']
3
4  IF vm.hypervisor = "proxmox"
5  THEN
```

```
6  status, message, params <- Proxmox_Library().
       resize_vm(params, params['vm']['hosts'])
7  ELSE IF vm.hypervisor = "vmware"
8  THEN
9  status, message, params <- Vsphere_Library().
       resize_vm(params, vm.hosts)
10 ENDIF
11
12 RETURN status, message, params
13 ENDFUNCTION
```

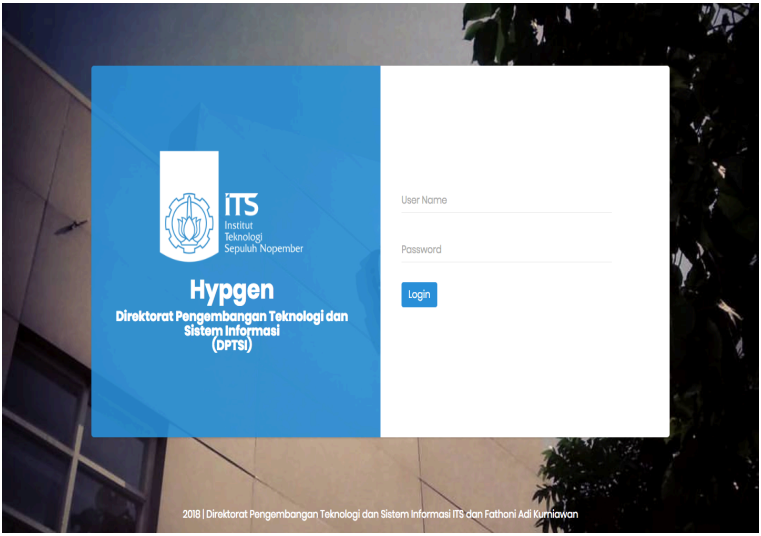
Kode Sumber IV.15: *Pseudocode* Fungsi *resize_vm* pada pada *class Hypervisor_Library*

4.3 Implementasi *Interface* Web

Untuk mengakses sistem, terdapat dua buah *interface* yang bisa digunakan yaitu *interface* web dan *command line interface*. *Interface* web dapat diakses melalui *port 9090*.

4.3.1 Implementasi Autentifikasi dan Otorisasi pada *Interface* Web

Untuk melakukan autentifikasi dan otorisasi pada *interface* web, pengguna mengisi *form username* dan *password* pada halaman *login*. Data *username* dan *password* dikirim ke *end-point middleware*. *Middleware* akan mengirim umpan balik berupa *token* yang digunakan untuk autentifikasi dan otorisasi. Selanjutnya *token* akan disimpan pada *cookie*.Tampilan halaman *login* dapat dilihat pada Gambar 4.2



Gambar 4.2: Implementasi Halaman Login pada *Interface* Web

Token akan dikirim oleh *interface* web pada *header* permintaan dari pengguna saat pengguna mengakses halaman pada *interface* web.

4.3.2 Implementasi *End-point* pada *Interface* Web

Untuk mengakses *interface* web, penulis menyediakan *end-point* yang dapat digunakan untuk mengatur sistem maupun untuk melakukan manajemen pada *virtual machine*.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	*End-point*	*Method*	*Keterangan*
1	`/`	GET	Merupakan halaman *root*, ketika pengguna mengakses *end-point* maka akan dialihkan ke `/login`.
2	`/dashboard`	GET	Merupakan halaman dasbor.
3	`/host`	GET	Halaman untuk melihat *host* yang tersedia.
4	`/host/create`	POST	*End-point* untuk memproses *form* data *host* baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
5	`/host/create`	GET	Halaman mengisi *form* data *host* baru.
6	`/host/<hypervisor_id>/<host_id>`	DELETE	*End-point* untuk menghapus *host* terpilih. *Interface* web akan mengirimkan *vm_id* dan *hypervisor_id* dari hasil penguraian *end-point* ke *middleware*.
7	`/host/<hypervisor_id>/<host_id>`	PUT	*End-point* untuk memproses perubahan data *host* terpilih. *Interface* web akan mengirimkan data ke *middleware*.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	*End-point*	*Method*	*Keterangan*
8	`/host/ <hypervisor_ id>/<host_id> /edit`	GET	Halaman untuk mengisi *form* perubahan data *host*.
9	`/login`	GET	Halaman login.
10	`/login`	POST	*End-point* untuk memproses permintaan *login* pengguna, Pada *end-point* ini, *interface* web akan mengirimkan data *username* dan *password* ke *middleware* untuk autentifikasi dan otorisasi.
11	`/logout`	GET	*End-point* untuk keluat dari sistem.
12	`/os`	POST	*End-point* untuk memproses *form* data versi sistem operasi baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
13	`/os`	GET	Halaman untuk melihat versi sistem operasi yang tersedia.
14	`/os/create`	GET	Halaman mengisi *form* data versi sistem operasi baru.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	***End-point***	***Method***	***Keterangan***
15	`/os/<os_id>`	DELETE	*End-point* untuk menghapus versi sistem operasi terpilih. *Interface* web akan mengirimkan *os_id* dari hasil penguraian *end-point* ke *middleware*.
16	`/os/<os_id>`	PUT	*End-point* untuk memproses perubahan data versi sistem operasi terpilih. *Interface* web akan mengirimkan data ke *middleware*.
17	`/os/<os_id> /edit`	GET	Halaman untuk mengisi *form* perubahan data versi sistem operasi.
18	`/resource`	GET	Halaman untuk melihat kategori *resource* yang tersedia.
19	`/resource`	POST	*End-point* untuk memproses data kategori *resource* baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
20	`/resource/ create`	GET	Halaman mengisi *form* data kategori *resource* baru.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	***End-point***	***Method***	***Keterangan***
21	`/resource/ <resource_id>`	DELETE	*End-point* untuk menghapus kategori *resource* terpilih. *Interface* web akan mengirimkan *resource_id* dari hasil penguraian *end-point* ke *middleware*.
22	`/resource/ <resource_id>`	PUT	*End-point* untuk memproses perubahan data kategori *resource* terpilih. *Interface* web akan mengirimkan data ke *middleware*.
23	`/resource/ <resource_id> /edit`	GET	Halaman untuk mengisi kategori *resource* perubahan data *host*.
24	`/task`	GET	Halaman untuk melihat riwayat *task*.
25	`/template`	GET	Halaman untuk melihat *template* sistem operasi yang tersedia.
26	`/template/ create`	GET	Halaman mengisi *form* data *template* sistem operas barui.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	***End-point***	***Method***	***Keterangan***
27	`/template/ create`	POST	*End-point* untuk memproses data versi sistem operasi baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
28	`/template/ <hypervisor_ id> /<template_ id>`	PUT	*End-point* untuk memproses perubahan data *template* sistem operasi terpilih. *Interface* web akan mengirimkan data ke *middleware*.
28	`/template/ <hypervisor_ id> /<template_ id>`	DELETE	*End-point* untuk menghapus *template* sistem operasi terpilih. *Interface* web akan mengirimkan *template_id* dan *hypervisor_id* dari hasil penguraian *end-point* ke *middleware*.
29	`/template/ <hypervisor_ id> /<template_ id>/edit`	GET	Halaman untuk mengisi *form* perubahan data *template* sistem operasi.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	***End-point***	***Method***	***Keterangan***
30	`/token`	POST	*End-point* untuk memproses data *API Secret Key* baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
31	`/token`	GET	Halaman untuk melihat *API Secret Key* yang tersedia.
32	`/token/create`	GET	Halaman mengisi *form* data *API Secret Key* versi sistem operasi baru.
33	`/token/ <token_id>`	DELETE	*End-point* untuk menghapus *API Secret Key* terpilih. *Interface* web akan mengirimkan *token_id* dari hasil penguraian *end-point* ke *middleware*.
34	`/token/ <token_id>`	PUT	*End-point* untuk memproses perubahan data *API Secret Key* terpilih. *Interface* web akan mengirimkan data ke *middleware*.
35	`/token/ <token_id> /edit`	GET	Halaman untuk mengisi *form* perubahan data *API Secret Key*.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	***End-point***	***Method***	***Keterangan***
36	`/user`	GET	Halaman untuk melihat pengguna yang terdaftar pada sistem.
37	`/user`	POST	*End-point* untuk memproses data pengguna baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
38	`/user/create`	GET	Halaman mendaftarkan pengguna pada sistem.
39	`/user/<user_id>`	PUT	*End-point* untuk memproses perubahan data pengguna terpilih. *Interface* web akan mengirimkan data ke *middleware*.
40	`/user/<user_id>`	DELETE	*End-point* untuk menghapus pengguna terpilih. *Interface* web akan mengirimkan *user_id* dari hasil penguraian *end-point* ke *middleware*.
41	`/user/<user_id>/edit`	GET	Halaman untuk mengisi *form* perubahan data pengguna.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	***End-point***	***Method***	***Keterangan***
42	`/vm`	POST	*End-point* untuk memproses data *virtual machine* baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
43	`/vm/create`	GET	Halaman untuk membuat *virtual machine* baru.
44	`/vm/<vm_id> /destory`	GET	Halaman konfirmasi penghapusan *virtual machine*.
45	`/vm/<vm_id> /history`	GET	Halaman untuk melihat riwayat *task virtual machine*.
46	`/vm/<vm_id> /owner`	GET	Halaman melihat pemilik *virtual machine*.
47	`/vm/<vm_id> /owner`	POST	*End-point* untuk memproses data pemilik *virtual machine* baru dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	*End-point*	*Method*	*Keterangan*
48	`/vm/<vm_id> /owner/revoke`	POST	*End-point* untuk memproses penghapusan pemilik *virtual machine* dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
49	`/vm/<vm_id> /owner/revoke`	GET	Halaman untuk memilih pengguna yang akan dihapus dari kepemilikan *virtual machine*.
50	`/vm/<vm_id> /resize`	GET	Halaman untuk memilih perubahan kategori *resource* pada *virtual machine*.
51	`/vm/<vm_id> /resize`	POST	*End-point* untuk memproses perubahan *resource virtual machine* dan pada *end-point* ini, *interface* web akan mengirimkan data tersebut ke *middleware*.
52	`/vm/<vm_id> /start`	POST	*End-point* untuk menyalakan *virtual machine*.
53	`/vm/<vm_id> /stop`	POST	*End-point* untuk mematikan *virtual machine*.
54	`/vm/<vm_id>`	GET	Halaman untuk melihat data *virtual machine*.

Tabel 4.21: Tabel *End-point* pada *Interface* Web

No	*End-point*	*Method*	*Keterangan*
55	`/vm/<vm_id>`	DELETE	*End-point* untuk menghapus *virtual machine* terpilih. *Interface* web akan mengirimkan *vm_id* dari hasil penguraian *end-point* ke *middleware*.

4.4 Implementasi *Command Line Interface*

Selain dapat diakses melalui *interface* web, sistem juga dapat diakses mennggunakan *Command Line Interface*. *Command Line Interface* dibangun menggunakan bahasa pemrograman Python.

4.4.1 Implementasi Autentifikasi dan Otorisasi pada *Command Line Interface*

Untuk mengakses sistem melalui *Command Line Interface*, pengguna harus membuat *API Secret Key* pada *interface* web. *API Secret Key* barus diatur terlebih dahulu pada aplikasi. Ketika pengguna menjalankan aplikasi dengan perintah tertentu, *API Secret Key* akan dikirim ke *middleware* terlebih dahulu untuk mendapatkan *token*. *API Secret Key* juga membatasi hak akses dari pengguna. Terdapat dua kategori yaitu *write* dan *read only*. Untuk mengatur *API Secret Key* dapat dilihat pada Kode Sumber IV.16.

```
hypgen auth <API Secret Key>
```

Kode Sumber IV.16: Perintah Untuk Mengatur *API Secret Key*

4.4.2 Implementasi Manajemen *Virtual Machine* pada *Command Line Interface*

Pada aplikasi *Command Line Interface* terdapat *parameter* untuk melakukan manajemen pada sistem. *Parameter* yang tersedia dapat dilihat pada Tabel 4.22.

Tabel 4.22: Tabel *Parametert* pada *Command Line Interface*

No	***Parameter***	***Keterangan***
1	*hypgen ps*	Untuk melihat status *virtual machine*.
2	*hypgen config*	Untuk mengatur dan melihat konfigurasi *Command Line Interface*. Pada *parameter* ini, pengguna dapat mengatur alamat *HTTP Rest API*
3	*hypgen show resource*	Untuk melihat kategori *resource* yang tersedia.
4	*hypgen show resource*	Untuk melihat kategori *resource* yang tersedia.
5	*hypgen vm add*	Untuk membuat *virtual machine* baru.
6	*hypgen vm rm <vm_id>*	Untuk menghapus *virtual machine* tertentu.
7	*hypgen vm start <vm_id>*	Untuk menyalakan *virtual machine* tertentu.
7	*hypgen vm start <vm_id>*	Untuk mematikan *virtual machine* tertentu.

BAB V

PENGUJIAN DAN EVALUASI

5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari: satu server *middleware*, dua server *proxmox*, satu server *Vmware ESXI*, satu server menggunakan Windows Server 2016 sebagai *Vmware Vcenter* dan Komputer Penguji. Untuk melakukan pengujian performa, penulis membuat script Python untuk melakukan permintaan sejumlah skenario pengujian.

Spesifikasi untuk setiap komponen yang digunakan ditunjukkan pada Tabel 5.1.

Tabel 5.1: Spesifikasi Komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
1	Middleware	4 core processor, 4GB RAM	Python 2.7
2	Proxmox	4 core processor, 8GB RAM	*Hypervisor* Proxmox
3	Proxmox	4 core processor, 8GB RAM	*Hypervisor* Proxmox
4	VMware ESXI	4 core processor, 4GB RAM	*Vmware ESXI*
5	Windows Server	8 core processor, 16GB RAM	Windows Server 2016 dan *Vmware Vcenter for Windows*
6	Komputer penguji	4 core processor, 4GB RAM	MacOS High Sierra, Insomnia, Python 2.7

5.2 Skenario Uji Coba

Uji coba akan dilakukan untuk mengetahui keberhasilan sistem yang telah dibangun. Skenario pengujian dibedakan menjadi 2 bagian, yaitu:

- **Uji Fungsionalitas**
  Pengujian ini didasrkan pada fungsionalitas yang disajikan sistem.
- **Uji Performa**
  Pengujian ini untuk menguji ketahanan sistem terhadap sejumlah permintaan ke aplikasi secara bersamaan sejumlah pengguna yang meminta *virtual machine* baru. Pengujian dilakukan dengan melakukan *benchmark* pada sistem.

5.2.1 Skenario Uji Coba Fungsionalitas

Uji fungsionalitas dibagi menjadi 3, yaitu uji mengelola sistem menggunakan *Rest Client* untuk mengakses *HTTP Rest API* secara langsung, mengelola sistem menggunakan *interface* web, dan mengelola *virtual machien* menggunakan *Command Line Interface*.

5.2.1.1 Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

Pada *middleware* terdapat *HTTP Rest API* yang menjadi gerbang untuk mengakses sistem. Pengujian dilakukan dengan menggunakan aplikasi *Rest Client* dengan cara mengakses *(*end-point) serta mengirimkan *parameter* yang dibutuhkan pada *HTTP Rest API* secara langsung. Rancangan pengujian dan hasil yang diharapkan ditunjukkan pada Tabel 5.2.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
1	Autentifikasi	Melakukan permintaan autentifikasi dan otorisasi dengan mengirimkan *username* dan *password*.	*HTTP Rest API* dapat mengirimkan umpan balik berupa *token*.
		Melakukan permintaan autentifikasi dan otorisasi dengan mengirimkan *HTTP Rest API*.	*HTTP Rest API* dapat mengirimkan umpan balik berupa *token*.
2	*Host*	Menambahkan server baru.	Data server baru dapat disimpan pada basis data sistem.
		Melihat daftar server yang tersedia.	Pengguna dapat melihat daftar server yang tersedia pada sistem.
		Melihat data server terpilih.	Pengguna dapat melihat detail data server yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
		Memperbaharui data server terpilih.	Pengguna dapat melakukan perubahan data pada server yang dipilih.
		Menghapus data server terpilih.	Pengguna menghapus data server yang dipilih.
3	*OS*	Menambahkan versi sistem operasi yang didukung oleh sistem.	Data versi sistem operasi baru dapat disimpan pada basis data sistem.
		Melihat daftar versi sistem operasi yang tersedia	Pengguna dapat melihat daftar versi sistem operasi yang tersedia pada sistem.
		Melihat data versi sistem operasi terpilih.	Pengguna dapat melihat detail data versi sistem operasi yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
		Memperbaharui data versi sistem operasi terpilih.	Pengguna dapat melakukan perubahan data pada versi sistem operasi yang dipilih.
		Menghapus data versi sistem operasi terpilih.	Pengguna menghapus data versi sistem operasi yang dipilih.
4	*Template*	Menambahkan *template* sistem operasi yang didukung oleh sistem.	Data *template* sistem operasi baru dapat disimpan pada basis data sistem.
		Melihat daftar *template* sistem operasi yang tersedia.	Pengguna dapat melihat daftar *template* sistem operasi yang tersedia pada sistem.
		Melihat data *template* sistem operasi terpilih.	Pengguna dapat melihat detail data *template* sistem operasi yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
		Memperbaharui data *template* sistem operasi terpilih.	Pengguna dapat melakukan perubahan data pada *template* sistem operasi yang dipilih.
		Menghapus data *template* sistem operasi terpilih.	Pengguna menghapus data *template* sistem operasi yang dipilih.
5	Kategori *Resource*	Menambahkan kategori *resource* untuk pilihan *resource virtual machine*.	Data kategori *resource* baru dapat disimpan pada basis data sistem.
		Melihat daftar kategori *resource* yang tersedia.	Pengguna dapat melihat daftar kategori *resource* yang tersedia pada sistem.
		Melihat data kategori *resource* terpilih.	Pengguna dapat melihat detail kategori *resource* yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
		Memperbaharui data kategori *resource* terpilih.	Pengguna dapat melakukan perubahan data pada kategori *resource* yang dipilih.
		Menghapus data kategori *resource* terpilih.	Pengguna menghapus data kategori *resource* yang dipilih.
6	*User*	Mendaftarkan pengguna pada sistem.	Data pengguna baru dapat disimpan pada basis data sistem.
		Melihat daftar pengguna yang terdaftar pada sistem.	Pengguna dapat melihat daftar pengguna yang terdaftar pada sistem.
		Melihat data pengguna terpilih.	Pengguna dapat melihat detail pengguna yang dipilih.
		Memperbaharui data pengguna terpilih.	Pengguna dapat melakukan perubahan data pada pengguna yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
		Menghapus data pengguna terpilih.	Pengguna menghapus data pengguna yang dipilih.
7	*API Secret Key*	Membuat *API Secret Key* baru.	Data *API Secret Key* baru dapat disimpan pada basis data sistem.
		Melihat daftar *API Secret Key* yang tersedia pada pengguna tertentu.	Pengguna dapat melihat daftar *API Secret Key* yang tersedia.
		Melihat data *API Secret Key* terpilih.	Pengguna dapat melihat detail *API Secret Key* yang dipilih.
		Memperbaharui data *API Secret Key* terpilih.	Pengguna dapat melakukan perubahan data pada *API Secret Key* yang dipilih.
		Menghapus data *API Secret Key* terpilih.	Pengguna menghapus data *API Secret Key* yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
8	*Virtual Machine*	Membuat *virtual machine* baru.	Data *virtual machine* baru dibuat dan data terkait *virtual machine* tersimpan pada basis data sistem.
		Melihat daftar *virtual machine* yang tersedia pada pengguna tertentu.	Pengguna dapat melihat daftar *virtual machine* yang tersedia.
		Melihat detail data *virtual machine* terpilih.	Pengguna dapat melihat detail data *virtual machine* yang dipilih.
		Mengubah kategori *resource virtual machine* terpilih.	Pengguna dapat melakukan perubahan data pada kategori *resource virtual machine* yang dipilih.
		Menghapus *virtual machine* terpilih.	Pengguna dapat menghapus *virtual machine* yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
		Mematikan *virtual machine* terpilih.	Pengguna dapat mematikan *virtual machine* yang dipilih.
		Menyalakan *virtual machine* terpilih.	Pengguna dapat menyalakan *virtual machine* yang dipilih.
		Membagikan hak pengelolaan *virtual machine* kepada pengguna lain.	Pengguna dapat membagikan hak pengelolaan *virtual machine* kepada pengguna yang dipilih.
		Melihat daftar pengguna yang memiliki hak pengelolaan *virtual machine*.	Pengguna dapat melihat daftar pengguna yang memiliki hak pengelolaan *virtual machine*.
		Menghapus pengguna terpilih yang dari hak pengelolaan *virtual machine*.	Pengguna dapat menghapus pengguna terpilih dari hak pengelolaan *virtual machine*.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil Harapan
		Melihat riwayat *task* yang dilakukan pengguna terhadap *virtual machine*.	Pengguna dapat melihat daftar riwayat *task* yang dilakukan dirinya sendiri maupun pengguna lain terhadap *virtual machine*.
		Melihat status *virtual machine* terpilih.	Pengguna dapat melihat status *virtual machine* yang dipilih.
9	*Task*	Melihat daftar *task*	Pengguna dapat melihat daftar *task* yang dilakukan oleh pengguna yang sedang *login*

5.2.1.2 Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

Pengujian fungsionalitas selanjutnya, penulis menguji fitur-fitur pengelolaan sistem melalui *interface* web. Rancangan pengujian dan hasil yang diharapkan ditunjukkan pada Tabel 5.3.

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
1	Autentifikasi	Melakukan permintaan autentifikasi dan otorisasi dengan mengirimkan *username* dan *password*.	Pengguna dapat masuk pada sistem dan melihat halaman dasbor.
2	*Host*	Menambahkan server baru.	Data server baru dapat diteruskan oleh *interface* web ke middleware dan disimpan pada basis data sistem.
		Melihat daftar server yang tersedia.	Pengguna dapat melihat daftar server yang tersedia pada sistem.
		Memperbaharui data server terpilih.	Pengguna dapat melakukan perubahan data pada server yang dipilih.
		Menghapus data server terpilih.	Pengguna dapat menghapus data server yang dipilih.

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
3	*OS*	Menambahkan versi sistem operasi yang didukung oleh sistem.	Data versi sistem operasi baru dapat disimpan pada basis data sistem.
		Melihat daftar versi sistem operasi yang tersedia	Pengguna dapat melihat daftar versi sistem operasi yang tersedia pada sistem.
		Memperbaharui data versi sistem operasi terpilih.	Pengguna dapat melakukan perubahan data pada versi sistem operasi yang dipilih.
		Menghapus data versi sistem operasi terpilih.	Pengguna dapat menghapus data versi sistem operasi yang dipilih.
4	*Template*	Menambahkan *template* sistem operasi yang didukung oleh sistem.	Data *template* sistem operasi baru dapat disimpan pada basis data sistem.

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
		Melihat daftar *template* sistem operasi yang tersedia.	Pengguna dapat melihat daftar *template* sistem operasi yang tersedia pada sistem.
		Melihat data *template* sistem operasi terpilih.	Pengguna dapat melihat detail data *template* sistem operasi yang dipilih.
		Memperbaharui data *template* sistem operasi terpilih.	Pengguna dapat melakukan perubahan data pada *template* sistem operasi yang dipilih.
		Menghapus data *template* sistem operasi terpilih.	Pengguna dapat menghapus data *template* sistem operasi yang dipilih.
5	Kategori *Resource*	Menambahkan kategori *resource* untuk pilihan *resource virtual machine*.	Data kategori *resource* baru dapat disimpan pada basis data sistem.

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
		Melihat daftar kategori *resource* yang tersedia.	Pengguna dapat melihat daftar kategori *resource* yang tersedia pada sistem.
		Memperbaharui data kategori *resource* terpilih.	Pengguna dapat melakukan perubahan data pada kategori *resource* yang dipilih.
		Menghapus data kategori *resource* terpilih.	Pengguna data menghapus data kategori *resource* yang dipilih.
6	*User*	Mendaftarkan pengguna pada sistem.	Data pengguna baru dapat disimpan pada basis data sistem.
		Melihat daftar pengguna yang terdaftar pada sistem.	Pengguna dapat melihat daftar pengguna yang terdaftar pada sistem.
		Memperbaharui data pengguna terpilih.	Pengguna dapat melakukan perubahan data pada pengguna yang dipilih.

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
		Menghapus data pengguna terpilih.	Pengguna dapat menghapus data pengguna yang dipilih.
7	*API Secret Key*	Membuat *API Secret Key* baru.	Data *API Secret Key* baru dapat disimpan pada basis data sistem.
		Melihat daftar *API Secret Key* yang tersedia pada pengguna tertentu.	Pengguna dapat melihat daftar *API Secret Key* yang tersedia.
		Memperbaharui data *API Secret Key* terpilih.	Pengguna dapat melakukan perubahan data pada *API Secret Key* yang dipilih.
		Menghapus data *API Secret Key* terpilih.	Pengguna menghapus data *API Secret Key* yang dipilih.
8	*Virtual Machine*	Membuat *virtual machine* baru.	Data *virtual machine* baru dibuat dan data terkait *virtual machine* tersimpan pada basis data sistem.

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
		Melihat detail data *virtual machine* terpilih.	Pengguna dapat melihat detail data *virtual machine* yang dipilih.
		Mengubah kategori *resource virtual machine* terpilih.	Pengguna dapat melakukan perubahan data pada kategori *resource virtual machine* yang dipilih.
		Menghapus *virtual machine* terpilih.	Pengguna dapat menghapus *virtual machine* yang dipilih.
		Mematikan *virtual machine* terpilih.	Pengguna dapat mematikan *virtual machine* yang dipilih.
		Menyalakan *virtual machine* terpilih.	Pengguna dapat menyalakan *virtual machine* yang dipilih.
		Membagikan hak pengelolaan *virtual machine* kepada pengguna lain.	Pengguna dapat membagikan hak pengelolaan *virtual machine* kepada pengguna yang dipilih.

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
		Melihat daftar pengguna yang memiliki hak pengelolaan *virtual machine*.	Pengguna dapat melihat daftar pengguna yang memiliki hak pengelolaan *virtual machine*.
		Menghapus pengguna terpilih yang dari hak pengelolaan *virtual machine*.	Pengguna dapat menghapus pengguna terpilih dari hak pengelolaan *virtual machine*.
		Melihat riwayat *task* yang dilakukan pengguna terhadap *virtual machine*.	Pengguna dapat melihat daftar riwayat *task* yang dilakukan dirinya sendiri maupun pengguna lain terhadap *virtual machine*.
		Melihat status *virtual machine* terpilih.	Pengguna dapat melihat status *virtual machine* yang dipilih.
9	*Task*	Melihat daftar *task*	Pengguna dapat melihat daftar *task* yang dilakukan oleh pengguna yang sedang *login*

Tabel 5.3: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil Harapan
10	*Dashboard*	Melihat daftar *virtual machine* yang tersedia	Pengguna dapat melihat daftar *virtual machine* yang tersedia.

5.2.1.3 Uji Fungsionalitas Mengelola *Virtual Machine* Menggunakan *Command Line Interface*

Pengujian fungsionalitas selanjutnya, penulis menguji fitur-fitur pengelolaan *virtual machine* melalui *Command Line Interface*. Rancangan pengujian dan hasil yang diharapkan ditunjukkan pada Tabel 5.4.

Tabel 5.4: Skenario Uji Fungsionalitas Mengelola *Virtual Machine* Menggunakan *Command Line Interface*

No	Menu	Uji Coba	Hasil Harapan
1	Autentifikasi	Mengatur *API Secret Key* pada *Command Line Interface*	Pengguna dapat mengatur *API Secret Key* agar terautentifikasi dan terotentifikasi *Command Line Interface*.
2	Config	Mengatur konfigurasi *Command Line Interface*	Pengguna dapat mengatur konfigurasi dasar *Command Line Interface*.

Tabel 5.4: Skenario Uji Fungsionalitas Mengelola *Virtual Machine* Menggunakan *Command Line Interface*

No	Menu	Uji Coba	Hasil Harapan
3	Show	Melihat daftar versi sistem operasi yang tersedia	Pengguna dapat melihat daftar versi sistem operasi yang tersedia.
		Melihat kategori *resource* yang tersedia	Pengguna dapat melihat kategori *resource* yang tersedia.
4	Status	Melihat status *virtual machine*	Pengguna dapat melihat status *virtual machine*.
5	VM	Membuat *virtual machine* baru	Pengguna dapat melihat status *virtual machine*.
		Mematikan *virtual machine*	Pengguna dapat mematikan *virtual machine* melalui *Command Line Interface*.
		Menyalakan *virtual machine*	Pengguna dapat menyalakan *virtual machine* melalui *Command Line Interface*.

Tabel 5.4: Skenario Uji Fungsionalitas Mengelola *Virtual Machine* Menggunakan *Command Line Interface*

No	Menu	Uji Coba	Hasil Harapan
		Menghapus *virtual machine*	Pengguna dapat menghapus *virtual machine* melalui *Command Line Interface*.

5.2.1.4 Uji Fungsionalitas Distribusi Alokasi *Virtual Machine*

Pengujian fungsionalitas yang terakhir adalah pengujian distribusi alokasi *virtual machine* baru yang diminta oleh pengguna. Setiap permintaan alokasi *virtual machine* baru, sistem akan menghitung server terbaik untuk alokasi menggunakan algoritma AHP. Percobaan dilakukan dengan lima skenario jumlah *concurrent user* yang berbeda, yaitu sejumlah 1, 2, 3, 5, 10, 15 dan 20 pengguna. Pada pengujian ini akan dihitung berapa jumlah *virutal machine* yang dialokasikan pada server tertentu sesuai skenario yang sudah ditentukan.

5.2.2 Skenario Uji Coba Performa

Uji performa dilakukan dengan menggunakan *script* Python yang mensimulasikan permintaan pengguna untuk melakukan akses secara bersamaan ke aplikasi.*Script* Python akan mengakses *end-point* pembuatan *virtual machine* pada *HTTP Rest API* sejumlah pengguna yang ditentukan oleh penulis.

Percobaan dilakukan dengan lima skenario jumlah *concurrent user* yang berbeda, yaitu sejumlah 1, 2, 3, 5, 10, 15, dan 20 pengguna. Skenario tersebut sama debgab pengujian pada subbab 5.2.1.4. Pengujian *request* ini bertujuan untuk mengukur

kemampuan dari *middleware* dalam menangani permintaan alokasi *virtual machine* baru.

Keberhasilan permintaan dari pengguna, tidak hanya diukur dari waktu umpan balik dari *middleware* tetapi diukur juga kemampuan *middleware* dalam menangani *task* alokasi *virtual machine* baru yang diberikan pengguna sejumlah skenario yang sudah ditentukan. Pada pengujian performa, terdapat 2 pengujian yaitu pengujian terhadap kecapatan menganangi *request* dari pengguna dan pengujian terhadap keberhasilan *request* dari pengguna.

5.2.2.1 Uji Performa Kecepatan Menangani *Request*

Pengujian dilakukan dengan mengukur jumlah waktu yang diperlukan untuk menyelesaikan *request* yang dilakukan oleh komputer penguji. Waktu yang diukur adalah total waktu yang dibutuhkan dalam alokasi *virtual machine* pertama sampai dengan yang terakhir.

5.2.2.2 Uji Performa Keberhasilan *Request*

Pengujian dilakukan dengan menghitung jumlah *request* yang gagal dilakukan selama skenario dijalankan. Dari semua jumlah *request* yang dikirimkan selama pengujian, akan didapatkan persen *request* yang gagal dilakukan.

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang telah dijelaskan pada subbab 5.2.

5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang dibangun.

5.3.1.1 Uji Mengelola Sistem Menggunakan *Rest Client*

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1 dan pada Tabel 5.2. Hasil pengujian seperti tertera pada Tabel 5.5.

Tabel 5.5: Hasil Uji Coba Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil
1	Autentifikasi	Melakukan permintaan autentifikasi dan otorisasi dengan mengirimkan *username* dan *password*.	OK
		Melakukan permintaan autentifikasi dan otorisasi dengan mengirimkan *HTTP Rest API*.	OK
2	*Host*	Menambahkan server baru.	OK
		Melihat daftar server yang tersedia.	OK
		Melihat data server terpilih.	OK

Tabel 5.5: Hasil Uji Coba Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil
		Memperbaharui data server terpilih.	OK
		Menghapus data server terpilih.	OK
3	*OS*	Menambahkan versi sistem operasi yang didukung oleh sistem.	OK
		Melihat daftar versi sistem operasi yang tersedia	OK
		Melihat data versi sistem operasi terpilih.	OK
		Memperbaharui data versi sistem operasi terpilih.	OK
		Menghapus data versi sistem operasi terpilih.	OK
4	*Template*	Menambahkan *template* sistem operasi yang didukung oleh sistem.	OK
		Melihat daftar *template* sistem operasi yang tersedia.	OK

Tabel 5.5: Hasil Uji Coba Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil
		Melihat data *template* sistem operasi terpilih.	OK
		Memperbaharui data *template* sistem operasi terpilih.	OK
		Menghapus data *template* sistem operasi terpilih.	OK
5	Kategori *Resource*	Menambahkan kategori *resource* untuk pilihan *resource virtual machine*.	OK
		Melihat daftar kategori *resource* yang tersedia.	OK
		Melihat data kategori *resource* terpilih.	OK
		Memperbaharui data kategori *resource* terpilih.	OK
		Menghapus data kategori *resource* terpilih.	OK
6	*User*	Mendaftarkan pengguna pada sistem.	OK

Tabel 5.5: Hasil Uji Coba Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil
		Melihat daftar pengguna yang terdaftar pada sistem.	OK
		Melihat data pengguna terpilih.	OK
		Memperbaharui data pengguna terpilih.	OK
		Menghapus data pengguna terpilih.	OK
7	*API Secret Key*	Membuat *API Secret Key* baru.	OK
		Melihat daftar *API Secret Key* yang tersedia pada pengguna tertentu.	OK
		Melihat data *API Secret Key* terpilih.	OK
		Memperbaharui data *API Secret Key* terpilih.	OK
		Menghapus data *API Secret Key* terpilih.	OK
8	*Virtual Machine*	OK	
		Melihat daftar *virtual machine* yang tersedia pada pengguna tertentu.	OK

Tabel 5.5: Hasil Uji Coba Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil
		Melihat detail data *virtual machine* terpilih.	OK
		Mengubah kategori *resource virtual machine* terpilih.	OK
		Menghapus *virtual machine* terpilih.	OK
		Mematikan *virtual machine* terpilih.	OK
		Menyalakan *virtual machine* terpilih.	OK
		Membagikan hak pengelolaan *virtual machine* kepada pengguna lain.	OK
		Melihat daftar pengguna yang memiliki hak pengelolaan *virtual machine*.	OK .
		Menghapus pengguna terpilih yang dari hak pengelolaan *virtual machine*.	OK
		Melihat riwayat *task* yang dilakukan pengguna terhadap *virtual machine*.	OK

Tabel 5.5: Hasil Uji Coba Mengelola Sistem Menggunakan *Rest Client*

No	Menu	Uji Coba	Hasil
		Melihat status *virtual machine* terpilih.	OK
9	*Task*	Melihat daftar *task*	OK

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.2, hasil uji coba menunjukkan semua skenario berhasil ditangani.

5.3.1.2 Uji Fungsionalitas Mengelola Sistem Menggunakan *Interface* Web

Sesuai dengan skenario pengujian yang dilakukan pada *interface* web. Pengujian dilakukan dengan menguji setiap menu pada *interface* web. Hasil uji coba dapat dilihat pada Table 5.6. Semua skenario yang direncanakan berhasil ditangani.

Tabel 5.6: Hasil Uji Coba Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil
1	Autentifikasi	Melakukan permintaan autentifikasi dan otorisasi dengan mengirimkan *username* dan *password*.	OK
2	*Host*	Menambahkan server baru.	OK
		Melihat daftar server yang tersedia.	OK

Tabel 5.6: Hasil Uji Coba Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil
		Memperbaharui data server terpilih.	OK
		Menghapus data server terpilih.	OK
3	*OS*	Menambahkan versi sistem operasi yang didukung oleh sistem.	OK
		Melihat daftar versi sistem operasi yang tersedia	OK
		Memperbaharui data versi sistem operasi terpilih.	OK
		Menghapus data versi sistem operasi terpilih.	OK
4	*Template*	Menambahkan *template* sistem operasi yang didukung oleh sistem.	OK
		Melihat daftar *template* sistem operasi yang tersedia.	OK
		Melihat data *template* sistem operasi terpilih.	OK

Tabel 5.6: Hasil Uji Coba Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil
		Memperbaharui data *template* sistem operasi terpilih.	OK
		Menghapus data *template* sistem operasi terpilih.	OK
5	Kategori *Resource*	Menambahkan kategori *resource* untuk pilihan *resource virtual machine*.	OK
		Melihat daftar kategori *resource* yang tersedia.	OK
		Memperbaharui data kategori *resource* terpilih.	OK
		Menghapus data kategori *resource* terpilih.	OK
6	*User*	Mendaftarkan pengguna pada sistem.	OK
		Melihat daftar pengguna yang terdaftar pada sistem.	OK
		Memperbaharui data pengguna terpilih.	OK

Tabel 5.6: Hasil Uji Coba Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil
		Menghapus data pengguna terpilih.	OK
7	*API Secret Key*	Membuat *API Secret Key* baru.	OK
		Melihat daftar *API Secret Key* yang tersedia pada pengguna tertentu.	OK
		Memperbaharui data *API Secret Key* terpilih.	OK
		Menghapus data *API Secret Key* terpilih.	OK
8	*Virtual Machine*	Membuat *virtual machine* baru.	OK.
		Melihat detail data *virtual machine* terpilih.	OK
		Mengubah kategori *resource virtual machine* terpilih.	OK
		Menghapus *virtual machine* terpilih.	OK
		Mematikan *virtual machine* terpilih.	OK
		Menyalakan *virtual machine* terpilih.	OK

Tabel 5.6: Hasil Uji Coba Mengelola Sistem Menggunakan *Interface* Web

No	Menu	Uji Coba	Hasil
		Membagikan hak pengelolaan *virtual machine* kepada pengguna lain.	OK
		Melihat daftar pengguna yang memiliki hak pengelolaan *virtual machine*.	OK
		Menghapus pengguna terpilih yang dari hak pengelolaan *virtual machine*.	OK
		Melihat riwayat *task* yang dilakukan pengguna terhadap *virtual machine*.	OK
		Melihat status *virtual machine* terpilih.	OK
9	*Task*	Melihat daftar *task*	OK
10	*Dashboard*	Melihat daftar *virtual machine* yang tersedia	OK

5.3.1.3 Uji Fungsionalitas Mengelola *Virtual Machine* Menggunakan *Command Line Interface*

Sesuai dengan skenario pengujian yang dilakukan pada *Command Line Interface*. Pengujian dilakukan dengan menguji setiap parameter/menu pada *Command Line Interface*. Hasil uji coba dapat dilihat pada Table 5.7. Semua skenario yang direncanakan berhasil ditangani.

Tabel 5.7: Hasil Uji Coba Mengelola *Virtual Machine* Menggunakan *Command Line Interface*

No	Menu	Uji Coba	Hasil Harapan
1	Autentifikasi	Mengatur *API Secret Key* pada *Command Line Interface*	OK
2	Config	Mengatur konfigurasi *Command Line Interface*	OK
3	Show	Melihat daftar versi sistem operasi yang tersedia	OK
		Melihat kategori *resource* yang tersedia	OK
4	Status	Melihat status *virtual machine*	OK
5	VM	Membuat *virtual machine* baru	OK
		Mematikan *virtual machine*	OK
		Menyalakan *virtual machine*	OK

Tabel 5.7: Hasil Uji Coba Mengelola *Virtual Machine* Menggunakan *Command Line Interface*

No	Menu	Uji Coba	Hasil
		Menghapus *virtual machine*	OK

5.3.2 Hasil Uji Performa

Seperti yang sudah dijelaskan pada subbab 5.2 pengujian performa dilakukan dengan melakukan akses ke aplikasi dengan sejumlah pengguna secara bersama-sama. Pengujian dilaukan dengan memberikan *request* secara berkelanjutan dengan jumlah pengguna terdiri dari lima bagian, yaitu 800, 1600, 2400, 3200, dan 4000 pengguna. Untuk jumlah *request* yang dihasilkan dari masing-masing pengguna selama rentang waktu request $\pm$ 15 detik dapat dilihat pada Tabel 5.8. Jumlah tersebut akan diolah oleh *reactive model*. Lalu jumlah penggunaan CPU dan memory selama menangani *request* tersebut akan digunakan oleh *proactive model* untuk menambahkan atau mengurangi *container* yang ada.

Tabel 5.8: Jumlah *Request* ke Aplikasi

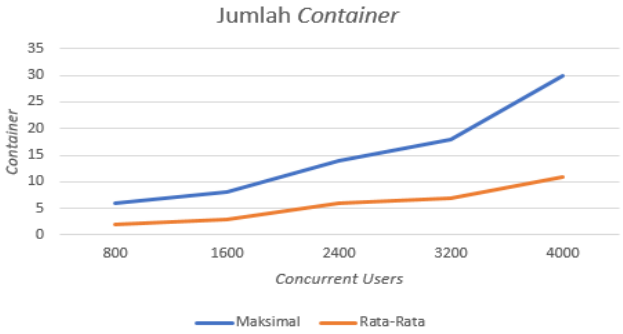
Concurrent Users	**Jumlah *Request***
800	$\pm$ 16.925
1.600	$\pm$ 26.650
2.400	$\pm$ 34.943
3.200	$\pm$ 50.092
4.000	$\pm$ 57.750

Pada Tabel 5.9 dapat dilihat jumlah *container* yang terbentuk selama proses *request* dari *user* yang dilakukan selama enam kali. Nilai yang ditampilkan berupa nilai rata-rata selama percobaan dibulatkan ke atas. Sistem dapat menyediakan *container* sesuai dengan jumlah *request* yang diberikan, semakin banyak *request* yang dilakukan, maka *container* yang disediakan akan semakin banyak. Nilai *container* tersebut didapatkan dari perhitungan *proactive model*. Selain melihat jumlah *request*, penentuan *container* yang dibentuk juga dari jumlah sumber daya yang digunakan *container* berdasarkan perhitungan

menggunakan *reactive model*. Pada Gambar 5.1 dapat dilihat grafik dari jumlah *container* yang terbentuk berdasarkan jumlah *request* yang dilakukan.

Tabel 5.9: Jumlah *Container*

Concurrent Users	**Maksimal *Container***	**Rata-rata *Container***
800	6	2
1.600	8	3
2.400	14	6
3.200	18	7
4.000	30	11



Gambar 5.1: Grafik Jumlah *Container*

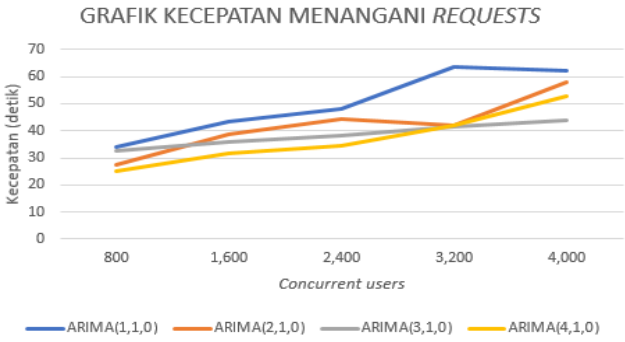
5.3.2.1 Kecepatan Menangani *Request*

Dari hasil uji coba kecepatan menangani *request*, dapat dilihat pada Table 5.10 dalam satuan detik bahwa semakin banyak *concurrent users*, semakin lama pula waktu yang diperlukan untuk menyeselaikannya. Request paling cepat ditangani dengan menggunakan prediksi ARIMA(4,1,0) dan

paling lambat menggunakan ARIMA(1,1,0). Hal tersebut terjadi karena kurang bagusnya hasil prediksi yang dihasilkan oleh ARIMA(1,1,0) yang mana kadang hasil prediksinya terlalu rendah atau terlalu tinggi. Dari hasil percobaan tersebut, dapat dilihat bahwa hampir semua *request* dapat ditangani di bawah satu menit. Lalu grafik hasil uji coba perhitungan kecepatan menangani *request* ditunjukkan pada Gambar 5.2.

Tabel 5.10: Kecepatan Menangani *Request*

	800	**1600**	**2400**	**3200**	**4000**
ARIMA(1,1,0)	34.167	43.286	48.143	63.857	62.286
ARIMA(2,1,0)	27.429	38.571	44.143	42.143	57.857
ARIMA(3,1,0)	32.429	36.000	38.429	41.571	43.857
ARIMA(4,1,0)	24.857	31.571	34.429	42.143	52.714



Gambar 5.2: Grafik Kecepatan Menangani *Request*

5.3.2.2 Penggunaan CPU

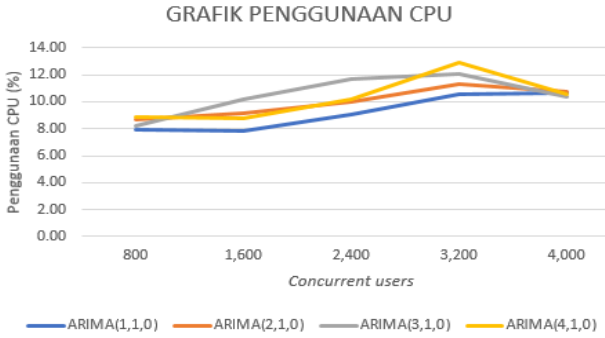
Dari hasil uji coba penggunaan CPU pada *server master host*, penggunaan CPU berada di bawah 15%. Penggunaan CPU yang diukur adalah penggunaan CPU yang dilakukan oleh *container*

dari aplikasi, tidak termasuk sistem. Jumlah *core* yang dimiliki oleh *processor* di *server master host* adalah 8 buah, yang artinya kurang lebih hanya satu core yang digunakan untuk menangani semua *request*. Hasil pengukuran penggunaan CPU dapat dilihat pada Tabel 5.11

Tabel 5.11: Penggunaan CPU

	800	**1600**	**2400**	**3200**	**4000**
ARIMA(1,1,0)	7.1%	7.8%	9.1%	10.5%	10.7%
ARIMA(2,1,0)	8.5%	9.2%	10.1%	11.3%	10.7%
ARIMA(3,1,0)	8.8%	10.2%	11.6%	12.1%	10.3%
ARIMA(4,1,0)	8.0%	8.3%	10.1%	12.9%	10.5%

Dari hasil uji coba, penggunaan prediksi yang berbeda tidak terlalu berpengaruh terhadap penggunaan CPU. Lalu, penggunaan CPU tergolong rendah, yaitu hanya sebesar ±10% untuk menangani semua *request* yang diberikan. Hasil uji coba performa penggunaan CPU ditunjukkan oleh dalam grafik pada Gambar 5.3.



Gambar 5.3: Grafik Penggunaan CPU

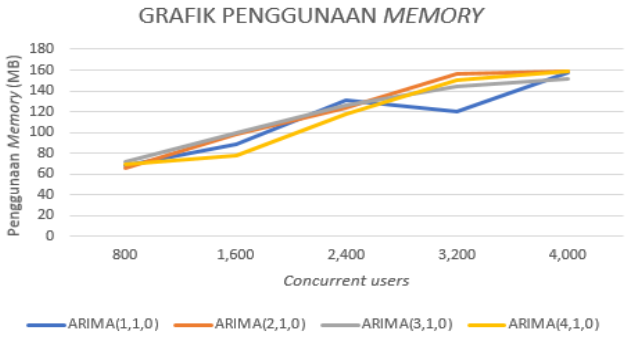
5.3.2.3 Penggunaan *Memory*

Dari hasil uji coba penggunaan *memory*, semakin banyak *request* yang diterima, semakin banyak *memory* yang diperlukan. Perhitungan penggunaan *memory* adalah rata-rata penggunaan dari masing-masing *container* sebuah aplikasi. Untuk masing-masing *container*, dibatasi penggunaan maksimal *memory* adalah 512 MB. Dari hasil uji coba ini, dapat dilihat pada Tabel 5.12 bahwa penggunaan terbesar hanya sebesar 158.71 MB. Artinya jumlah tersebut hanya menggunakan sepertiga dari keseluruhan *memory* yang bisa digunakan.

Tabel 5.12: Penggunaan *Memory*

	800	**1600**	**2400**	**3200**	**4000**
ARIMA(1,1,0)	67.91	88.97	130.79	120.14	157.73
ARIMA(2,1,0)	65.89	97.98	123.47	156.64	158.33
ARIMA(3,1,0)	72.20	99.72	125.56	144.42	152.14
ARIMA(4,1,0)	69.60	77.34	117.39	149.76	158.71

Hasil uji coba performa penggunaan *memory* dalam grafik ditunjukkan pada Gambar 5.4.



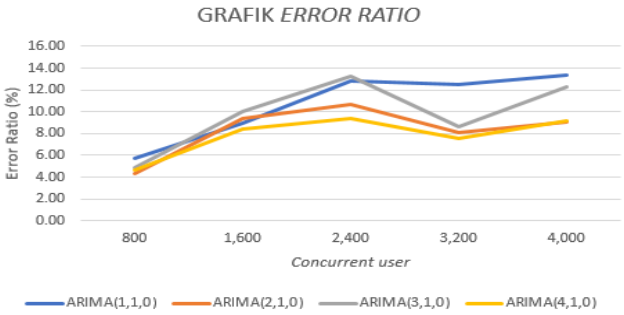
Gambar 5.4: Grafik Penggunaan Memory

5.3.2.4 Keberhasilan *Request*

Pada uji coba ini, dilakukan perhitungan seberapa besar jumlah *request* yang gagal dilakukan. Untuk jumlah *concurrent user* pada tingkat 800 dan 1600, dapat dilihat pada Table 5.13 *error* yang terjadi hampir sama. Prediksi menggunakan ARIMA(4,1,0) berhasil unggul karena menggunakan parameter yang lebih banyak. Namun hal tersebut tidak berlaku untuk ARIMA(3,1,0) karena walaupun parameternya lebih banyak dari ARIMA(2,1,0), tapi hasil prediksinya bisa meleset saat terjadi kondisi dimana koefisien negatif atau koefisien ke dua dikalikan dengan sebuah parameter bukan nol, dan koefisien lain dikalikan dengan parameter nol, maka hasil prediksinya akan negatif, yang mana seharusnya tidak mungkin ada *request* negatif.

Tabel 5.13: *Error Ratio Request*

	800	**1600**	**2400**	**3200**	**4000**
ARIMA(1,1,0)	5.72%	8.96%	12.85%	12.54%	13.38%
ARIMA(2,1,0)	4.31%	9.35%	10.68%	8.11%	9.04%
ARIMA(3,1,0)	4.84%	10.02%	13.22%	8.63%	12.24%
ARIMA(4,1,0)	4.62%	8.41%	9.39%	7.52%	9.21%



Gambar 5.5: Grafik Error Ratio

Dari uji coba itu, 90% lebih *request* berhasil ditangani. Hasil uji coba jumlah *request* yang gagal ditunjukkan dengan grafik pada Gambar 5.5.

(Halaman ini sengaja dikosongkan)

BAB VI

PENUTUP

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

6.1 Kesimpulan

Dari proses perencangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Sistem dapat menjalankan dan menyajikan satu atau lebih aplikasi web berbasis *docker* kepada *end-user* melalui domain yang disediakan.
2. Sistem dapat menyesuaikan sumber daya secara otomatis berdasarkan jumlah *request* dengan menggunakan *proactive model* dan penggunaan sumber daya, yaitu CPU dan *memory*, pada *container* dengan menggunakan *reactive model*.
3. Penggunaan *load balancer* cocok digunakan dengan aplikasi yang berjalan di atas *docker container*. Hal tersebut karena semua *request* ke aplikasi akan melalui *load balancer*. Jika terjadi penambahan dan pengurangan sumber daya, penyesuaian dengan cepat dilakukan dan hanya perlu merubah sedikit konfigurasi pada *load balancer* dan pengguna tidak perlu tahu apa yang terjadi di dalam sistem.
4. Prediksi jumlah *request* menggunakan ARIMA sudah bisa menangani skenario uji coba. Perbedaan *order* ARIMA yang digunakan mempengaruhi akurasi dalam menentukan *request* yang akan terjadi ke depannya. Dalam pengujian ini, ARIMA(4,1,0) memiliki hasil pengujian paling bagus dengan jumlah rata-rata *error request* yang paling rendah,

yaitu sebesar 7.83%. Lalu untuk kecepatan menerima *request*, ARIMA(2,1,0) dan ARIMA(4,1,0) memiliki konsistensi yang berbanding lurus dengan jumlah *request*.

5. Penggunaan sumber daya CPU dan *memory* tidak dipengaruhi oleh penggunaan ARIMA yang berbeda. Penggunaan sumber daya tersebut bergantung kepada jumlah *request*, semakin banyak *request* yang diberikan, penggunaan CPU dan *memory* akan semakin tinggi. Penggunaan CPU paling tinggi yaitu sebesar 12.9% dan penggunaan *memory* paling tinggi sebesar 158.71 MB. Dengan penggunaan tersebut, masih tersisa lebih dari setengah sumber daya yang bisa digunakan.
6. Sebuah *container* dari sebuah aplikasi dapat dibentuk dalam waktu ± 1 detik sehingga penambahan sumber daya bisa dilakukan dengan cepat dan proses untuk memperbarui konfigurasi dari HAProxy memerlukan waktu ± 5 detik. Selama proses tersebut, akses pengguna akan tertunda, namun tidak menunjukkan terjadinya *down*.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

1. Mengamankan komunikasi antar *server* karena saat ini *endpoint server* bisa diakses oleh siapapun. Hal tersebut bisa dilakukan dengan mengimplentasikan *private* IP dan menggunakan token untuk komunikasinya.
2. Pemodelan menggunakan ARIMA cukup baik, namun perlu dicoba untuk melakukan pembuatan model dengan *dataset* yang lebih baru. Selain itu, bisa mencoba alternatif pemodelan *time series* yang lain, seperti ARCH (Autoregressive Conditional Heteroskedasticity).

DAFTAR PUSTAKA

[1] N. Jain dan S. Choudhary, "Overview of Virtualization in Cloud Computing," in *Colossal Data Analysis and Networking (CDAN), Symposium on*, 2062.

[2] "General Python FAQ," 3 Mei 2018. [Daring]. Tersedia pada: https://docs.python.org/3/faq/general.html#what-is-python. [Diakses: 3 Mei 2018].

[3] "A simple framework for building complex web applications," 3 Mei 2018. [Daring]. Tersedia pada: https://pypi.org/project/Flask/. [Diakses: 3 Mei 2018].

[4] "Celery," 21 Mei 2018. [Daring]. Tersedia pada: http://www.celeryproject.org/. [Diakses: 21 Mei 2018].

[5] "Vmware vsphere Python SDK," 3 Mei 2018. [Daring]. Tersedia pada: https://pypi.org/project/pyvmomi/. [Diakses: 3 Mei 2018].

[6] "Python Wrapper for the Proxmox 2.x API (HTTP and SSH)," 3 Mei 2018. [Daring]. Tersedia pada: https://pypi.org/project/proxmoxer/. [Diakses: 3 Mei 2018].

[7] "What can PHP do?" 3 Mei 2018. [Daring]. Tersedia pada: https://secure.php.net/manual/en/intro-whatcando.php. [Diakses: 3 Mei 2018].

[8] "Redis," 10 April 2017. [Daring]. Tersedia pada: https://redis.io/. [Diakses: 10 April 2017].

[9] W. J. Gilmore, "Beginning PHP and MySQL From Novice to Professional," *Apress*, vol. 4th, hal. 477–480, 2010.

[10] F. Mohammad, V. Yadav, dan others, "Automatic decision making for multi-criteria load balancing in cloud environment using AHP," in *Computing, Communication & Automation (ICCCA), 2015 International Conference on*. IEEE, 2015, hal. 569–576.].

[11] T. L. Saaty, “Decision making with the analytic hierarchy process,” *International journal of services sciences*, vol. 1, no. 1, hal. 83–98, 2008.

[12] R. M. Ijtihadie, B. J. Santoso, D. Fablius, dan I. D. P. A. Nusawan, “Rancang bangun sistem penentuan keputusan untuk distribusi penyedian kontainer dengan multi kriteria secara dinamis,” 2017, hal. 198–199.].

LAMPIRAN A

INSTALASI PERANGKAT LUNAK

Instalasi Lingkungan Docker

Proses pemasangan Docker dpat dilakukan sesuai tahap berikut:

- Menambahkan repository Docker
  Langkah ini dilakukan untuk menambahkan *repository* Docker ke dalam paket `apt` agar dapat di unduh oleh Ubuntu. Untuk melakukannya, jalankan perintah berikut:

```
sudo apt-get -y install \
   apt-transport-https \
   ca-certificates \
   curl

curl -fsSL https://download.docker.com/linux/
    ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/
     linux/ubuntu \
    $ (lsb_release -cs) \
    stable"

sudo apt-get update
```

- Mengunduh Docker
  Docker dikembangkan dalam dua versi, yaitu CE (*Community Edition*) dan EE (*Enterprise Edition*). Dalam pengembangan sistem ini, digunakan Docker CE karena merupakan versi Docker yang gratis. Untuk mengunduh Docker CE, jalankan perintah `sudo apt-get -y install docker-ce`.

- Mencoba menjalankan Docker
  Untuk melakukan tes apakah Docker sudah terpasang dengan benar, gunakan perintah `sudo docker run hello-world`.

Instalasi Docker Registry

Docker Registry dikembangkan menggunakan Docker Compose. Dengan menggunakan Docker Compose, proses pemasangan Docker Registry menjadi lebih mudah dan fleksibel untuk dikembangkan ditempat lain. Docker Registry akan dijalankan pada satu *container* dan Nginx juga akan dijalankan di satu *container* lain yang berfungsi sebagai perantara komunikasi antara Docker Registry dengna dunia luar. Berikut adalah proses pengembangan Docker Registry yang penulis lakukan:

- Pemasangan Docker Compose
  $ `sudo apt-get -y install python-pip`
  $ `sudo pip install docker-compose`
- Pemasangan paket `apache2-utils`
  Pada paket `apache2-utils` terdapat fungsi `htpasswd` yang digunakan untuk membuat *hash password* untuk Nginx. Proses pemasangan paket dapat dilakukan dengan menjalankan perintah `sudo apt-get -y install apache2-utils`.
- Pemasangan dan pengaturan Docker Registry
  Buat folder `docker-registry` dan `data` dengan menjalankan perintah berikut:
  $ `mkdir /docker-registry && cd $_`
  $ `mkdir data`
  Folder `data` digunakan untuk menyimpan data yang dihasilkan dan digunakan oleh *container* Docker Registry. Kemudian di dalam folder `docker-registry` buat sebuah berkas dengan nama `docker-compose.yml` yang akan

digunakan oleh Docker Compose untuk membangun aplikasi. Tambahkan isi berkasnya sesuai dengan Kode Sumber A.1.

```
nginx:
image: "nginx:1.9"
ports:
  - 443:443
  - 80:80
links:
  - registry:registry
volumes:
  - ./nginx/:/etc/nginx/conf.d
registry:
  image: registry:2
  ports:
    - 127.0.0.1:5000:5000
  environment:
    REGISTRY_STORAGE_FILESYSTEM
        _ROOTDIRECTORY: /data
  volumes:
    - ./data:/data
    - ./registry/config.yml:/etc/docker
        /registry/config.yml
```

Kode Sumber A.1: Isi Berkas docker-compose.yml

- Pemasangan *container* Nginx Buat folder `nginx` di dalam folder `docker-registry`. Di dalam folder `nginx` buat berkas dengan nama `registry.conf` yang berfungsi sebagai berkas konfigurasi yang akan digunakan oleh Nginx. Isi berkas sesuai denga Kode Sumber A.2.

```
upstream docker-registry{
  server registry:5000;
}
```

```
server{
  listen 80;
  server_name registry.nota-no.life;
  return 301 https://
     $server_name$request_uri;
}
server{
  listen 443;
  server_name registry.nota-no.life;
  ssl on;
  ssl_certificate /etc/nginx/conf.d/
     cert.pem;
  ssl_certificate_key /etc/nginx/conf.d
     /privkey.pem;
  client_max_body_size 0;
  chunked_transfer_encoding on;
  location /v2/{
    if ($http_user_agent ~ "^(docker
       \/1\.(3|4|5(?!\.[0-9]-dev))|Go )
       .*$" ){
      return 404;
    }
    auth_basic "registry.localhost";
    auth_basic_user_file /etc/nginx/
       conf.d/registry.password;
    add_header 'Docker-Distribution-Api
       -Version' 'registry/2.0' always;
    proxy_pass http://docker-registry;
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP
       $remote_addr;
    proxy_set_header X-Forwarded-For
       $proxy_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto
            $scheme;
        proxy_read_timeout 900;
    }
}
```

Kode Sumber A.2: Isi Berkas registry.conf

Instalasi Pustaka Python

Dalam pengembangan sistem ini, digunakan berbagai pustaka pendukung. Pustaka pendukung yang digunakan merupakan pustaka untuk bahasa pemrograman Python. Berikut adalah daftar pustaka yang digunakan dan cara pemasangannya:

- Python Dev
  `$ sudo apt-get install python-dev`
- Flask
  `$ sudo pip install Flask`
- docker-py
  `$ sudo pip install docker`
- MySQLd
  `$ sudo apt-get install python-mysqldb`
- Redis
  `$ sudo pip install redis`
- RQ
  `$ sudo pip install rq`

Instalasi HAProxy

HAProxy dapat dipasang dengna mudah menggunakan `apt-get` karena perangkat lunak tersebut sudah tersedia pada *repository* Ubuntu. Untuk melakukan pemasangan HAProxy, gunakan perintah `apt-get install haproxy`.

Setelah HAProxy diunduh, perangkat lunak tersebut belum berjalan karena belum diaktifkan. Untuk mengaktifkan *service haproxy*, buka berkas di `/etc/default/harpoxy` kemudian ganti nilai `ENABLED` yang awalnya bernilai `0` menjadi `ENABLED=1`. Setelah itu service haproxy dapat dijalankan dengan menggunakan perintah `service harpoxy start`. Untuk konfigurasi dari HAProxy nantinya akan diurus oleh *confd*. *confd* akan menyesuaikan konfigurasi dari HAProxy sesuai dengan kebutuhan aplikasi yang tersedia.

Instalasi etcd dan confd

etcd dapat di unggah dengan menjalankan perintah berikut, `curl https://github.com/coreos/etcd/releases/download/v3.2.0-rc.0/etcd-v3.2.0-rc.0-linux-amd64.tar.gz`. Setelah proses unduh berhasil dilakukan, selanjutnya yang dilakukan adalah melakukan ekstrak berkasnya menggunakan perintah `tar -xvzf etcd-v3.2.0-rc.0-linux-amd64.tar.gz`. Berkas binary dari etcd bisa ditemukan pada folder `./bin/etcd`. Berkas inilah yang digunakan untuk menjalankan perangkat lunak etcd. Untuk menjalankannya, dapat dilakukan dengan menggunakan perintah `etcd --listen-client-urls http://0.0.0.0:5050 --advertise-client-urls http://128.199.250.137:5050`. Perintah tersebut memungkinkan etcd diakses oleh *host* lain dengan IP 128.199.250.137, yang merupakan host dari *load balancer* dan confd. Setelah proses tersebut, etcd sudah siap untuk digunakan.

Setelah etcd siap digunakan, selanjutnya adalah memasang confd. Untuk menginstall confd gunakan rangkaian perintah berikut:

```
$ mkdir -p $GOPATH/src/github.com/kelseyhightower
$    git clone https://github.com/kelseyhightower/
```

```
confd.git $GOPATH/src/github.com/kelseyhightower/
confd
$ cd $GOPATH/src/github.com/kelseyhightower/confd
$ ./build
```

Setelah berhasil memasang confd, selanjutnya buka berkas `/etc/confd/confd.toml` dan isi berkas sesuai dengan Kode Sumber A.3. Pengaturan tersebut bertujuan agar confd melakukan *listen* terhadap server etcd dan melakukan tindakan jika terjadi perubahan pada etcd.

```
confdir = "/etc/confd"
interval = 20
backend = "etcd"
nodes = [
        "http://128.199.250.137:5050"
]
prefix = "/"
scheme = "http"
verbose = true
```

Kode Sumber A.3: Isi Berkas confd.toml

Setelah melakukan konfigurasi confd, selanjutnya adalah membuat *template* konfigurasi untuk HAProxy. Buka berkas di `/etc/confd/templates/haproxy.cfg.tmpl`. Jika berkas tidak ada maka buat berkasnya dan isi berkas sesuai dengan Kode Sumber A.4.

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.
           sock mode 660 level admin
        stats timeout 30s
```

```
        daemon
defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors
           /400.http
        errorfile 403 /etc/haproxy/errors
           /403.http
        errorfile 408 /etc/haproxy/errors
           /408.http
        errorfile 500 /etc/haproxy/errors
           /500.http
        errorfile 502 /etc/haproxy/errors
           /502.http
        errorfile 503 /etc/haproxy/errors
           /503.http
        errorfile 504 /etc/haproxy/errors
           /504.http
frontend http-in
        bind *:80

        # Define hosts
        {{range gets "/images/*"}}
        {{$data := json .Value}}
                acl host_{{$data.image_name
                   }} hdr(host) -i {{$data.
                   domain}}.nota-no.life
        {{end}}
```

```
        ## Figure out which one to use
        {{range gets "/images/*"}}
        {{$data := json .Value}}
                use_backend {{$data.
                   image_name}}_cluster if
                   host_{{$data.image_name
                   }}
        {{end}}
{{range gets "/images/*"}}
{{$data := json .Value}}
backend {{$data.image_name}}_cluster
        mode http
        balance roundrobin
        option forwardfor
        cookie JSESSIONID prefix
        {{range $data.containers}}
        server {{.name}} {{.ip}}:{{.port}}
           check
        {{end}}
{{end}}
```

Kode Sumber A.4: Isi Berkas haproxy.cfg.tmpl

Langkah terakhir adalah membuat berkas konfigurasi untuk HAProxy di `/etc/confd/conf.d/haproxy.toml`. Jika berkas tidak ada, maka buat berkasnya dan isi berkas sesuai dengan Kode Sumber A.5.

```
[template]
src = "haproxy.cfg.tmpl"
dest = "/etc/haproxy/haproxy.cfg"
keys = [
        "/images"
]
```

```
reload_cmd = "iptables -I INPUT -p tcp --
    dport 80 --syn -j DROP && sleep 1 &&
    service haproxy restart && iptables -D
    INPUT -p tcp --dport 80 --syn -j DROP"
```

Kode Sumber A.5: Isi Berkas haproxy.toml

Setelah melakukan konfigurasi, selanjutnya adalah menjalankan confd dengan menggunakan perintah `confd &`.

Pemasangan Redis

Redis dapat dipasang dengan mempersiapkan kebutuhan pustaka pendukungnya. Pustaka yang digunakan adalah `build-essential` dan `tcl8.5`. Untuk melakukan pemasangannya, jalankan perintah berikut:

```
$ sudo apt-get install build-essential
$ sudo apt-get install tcl8.5
```

Setelah itu unduh aplikasi Redis dengan menjalankan perintah `wget http://download.redis.io/releases/redis-stable.tar.gz`. Setelah selesai diunduh, buka file dengan perintah berikut:

```
$ tar xzf redis-stable.tar.gz && cd redis-stable
```

Di dalam folder `redis-stable`, bangun Redis dari kode sumber dengan menjalankan perintah `make`. Setelah itu lakukan tes kode sumber dengan menjalankan `make test`. Setelah selesai, pasang Redis dengan menggunakan perinah `sudo make install`. Setelah selesai melakukan pemasangan, Redis dapat diaktifkan dengan menjalankan berkas bash dengan nama `install_server.sh`.

Untuk menambah pengaman pada Redis, diatur agar Redis hanya bisa dari *localhost*. Untuk melakukannya, buka file `/etc/redis/6379.conf`, kemudian cari baris `bind 127.0.0.1`. Hapus komen jika sebelumnya baris tersebut dalam

keadaan tidak aktif. Jika tidak ditemukan baris dengan isi tersebut, tambahkan pada akhir berkas baris tersebut.

Pemasangan kerangka kerja React

Pada pengembangan sistem ini, penggunaan pustaka React dibangun di atas konfigurasi Create React App. Untuk memasang Create React App, gunakan perintah `npm install -g create-react-app`. Setelah terpasang, untuk membangun aplikasinya jalankan perintah `create-react-app fe-controller`. Setelah proses tersebut, dasar dari aplikasi sudah terbangun dan siap untuk dikembangkan lebih lanjut.

(Halaman ini sengaja dikosongkan)

LAMPIRAN B

KODE SUMBER

Let's Encrypt Cross Signed

```
-----BEGIN CERTIFICATE-----
MIIEkjCCA3qgAwIBAgIQCgFBQgAAAVOF
    c2oLheynCDANBgkqhkiG9w0BAQsFADA/
MSQwIgYDVQQKExtEaWdpdGFsIFNpZ25h
    dHVyZSBUcnVzdCBDby4xFzAVBgNVBAMT
DkRTVCBSb290IENBIFgzMB4XDTE2MDMx
    NzE2NDA0NloXDTIxMDMxNzE2NDA0Nlow
SjELMAkGA1UEBhMCVVMxFjAUBgNVBAoT
    DUxldCdzIEVuY3J5cHQxIzAhBgNVBAMT
GkxldCdzIEVuY3J5cHQgQXV0aG9yaXR5
    IFgzMIIBIjANBgkqhkiG9w0BAQEFAAOC
AQ8AMIIBCgKCAQEAnNMM8FrlLke3cl03
    g7NoYzDq1zUmGSXhvb418XCSL7e4S0EF
q6meNQhY7LEqxGiHC6PjdeTm86dicbp5 gWAf15Gan/
    PQeGdxyGkOlZHP/uaZ6WA8
SMx+yk13EiSdRxta67nsHjcAHJyse6cF 6
    s5K671B5TaYucv9bTyWaN8jKkKQDIZ0
Z8h/pZq4UmEUEz9l6YKHy9v6Dlb2honz hT+Xhq+
    w3Brvaw2VFn3EK6BlspkENnWA
a6xK8xuQSXgvopZPKiAlKQTGdMDQMc2P
    MTiVFrqoM7hD8bEfwzB/onkxEz0tNvjj
/PIzark5McWvxI0NHWQWM6r6hCm21AvA 2
    H3DkwIDAQABo4IBfTCCAXkwEgYDVR0T
AQH/BAgwBgEB/wIBADAOBgNVHQ8BAf8E
    BAMCAYYwfwYIKwYBBQUHAQEEczBxMDIG
CCsGAQUFBzABhiZodHRwOi8vaXNyZy50
    cnVzdGlkLm9jc3AuaWRlbnRydXN0LmNv
bTA7BggrBgEFBQcwAoYvaHR0cDovL2Fw
    cHMuaWRlbnRydXN0LmNvbS9yb290cy9k
```

```
c3Ryb290Y2F4My5wN2MwHwYDVR0jBBgw
    FoAUxKexpHsscfrb4UuQdf/EFWCFiRAw
VAYDVR0gBE0wSzAIBgZngQwBAgEwPwYL
    KwYBBAGC3xMBAQEwMDAuBggrBgEFBQcC
ARYiaHR0cDovL2Nwcy5yb290LXgxLmxl
    dHNlbmNyeXB0Lm9yZzA8BgNVHR8ENTAz
MDGgL6AthitodHRwOi8vY3JsLmlkZW50
    cnVzdC5jb20vRFNUUk9PVENBWDNDUkwu
Y3JsMB0GA1UdDgQWBBSoSmpjBH3duubR
    ObemRWXv86jsoTANBgkqhkiG9w0BAQsF
AAOCAQEA3TPXEfNjWDjdGBX7CVW+dla5
    cEilaUcne8IkCJLxWh9KEik3JHRRHGJo
uM2VcGfl96S8TihRzZvoroed6ti6WqEB
    mtzw3Wodatg+VyOeph4EYpr/1wXKtx8/
wApIvJSwtmVi4MFU5aMqrSDE6ea73Mj2
    tcMyo5jMd6jmeWUHK8so/joWUoHOUgwu
X4Po1QYz+3dszkDqMp4fklxBwXRsW10K  XzPMTZ+
    sOPAveyxindmjkW8lGy+QsRlG
PfZ+G6Z6h7mjem0Y+iWlkYcV4PIWL1iw
    Bi8saCbGS5jN2p8M+X+Q7UNKEkROb3N6
KOqkqm57TH2H3eDJAkSnh6/DNFu0Qg==
−−−−−END  CERTIFICATE−−−−−
```

Kode Sumber B.1: Let's Encrypt X3 Cross Signed.pem

BIODATA PENULIS

Fathoni Adi Kurniawan, akbrab dipanggil Thoni lahir pada tanggal 4 Maret 1996 di kabupaten Klaten, Jawa Tengah. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Memiliki hobi antara lain mendengarkan musik dan mencoba *tool-tool* IT yang baru. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Media Informasi (Medfo) Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-2. Pernah menjadi staff National Programming Contest Schematics tahun 2015 dan dan pengembang web Schematics 2016. Selain itu penulis pernah menjadi asisten dosen di mata kuliah Sistem Operasi, Jaringan Komputer dan Komputasi Awan.