TUGAS AKHIR – KI091391

MANAJEMEN SUMBER DAYA UNTUK MENINGKATKAN EFISIENSI BEBAN KERJA KOMPUTASI AWAN MENGGUNAKAN KERANGKA KERJA OPENSTACK

AJI SETYO UTOMO
NRP 5110 100 010

Dosen Pembimbing I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.

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

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2014

FINAL PROJECT – KI091391

RESOURCE MANAGEMENT FOR IMPROVING CLOUD COMPUTING WORKLOAD EFFICIENCY USING OPENSTACK FRAMEWORK

AJI SETYO UTOMO
NRP 5110 100 010

Supervisor
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

INFORMATICS DEPARTMENT
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2013

LEMBAR PENGESAHAN

MANAJEMEN SUMBER DAYA UNTUK MENINGKATKAN EFISENSI BEBAN KERJA KOMPUTASI AWAN MENGGUNAKAN KERANGKA KERJA OPENSTACK

TUGAS AKHIR

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

Oleh:
AJI SETYO UTOMO
NRP. 5110 100 010

Disetujui oleh Pembimbing Tugas Akhir:

Waskitho Wibisono, S.Kom., M.Eng., Ph.D. .....
NIP: 197410222000031001 (Pembimbing 1)

Royyana Muslim I., S.Kom., M.Kom., Ph.D. .....
NIP: 197708242006041001 (Pembimbing 2)

SURABAYA
JULI, 2014

MANAJEMEN SUMBER DAYA UNTUK MENINGKATKAN EFISENSI BEBAN KERJA KOMPUTASI AWAN MENGGUNAKAN KERANGKA KERJA OPENSTACK

Nama Mahasiswa : Aji Setyo Utomo
NRP : 5110 100 010
Jurusan : Teknik Informatika FTIF-ITS
Dosen Pembimbing 1 : Waskitho Wibisono, S.Kom., M.Eng., Ph.D.
Dosen Pembimbing 2 : Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

ABSTRAK

Komputasi awan merupakan sebuah model komputasi pada sumber daya seperti *processor*, penyimpanan, dan jaringan yang diberikan sebagai layanan melalui jaringan intranet atupu internet. OpenStack merupakan salah satu perangkat lunak *open source* penyedia layanan komputasi awan yang berorientasi di bidang *Infrastucure as Service* (Iaas). OpenStack menjadi sangat popular dan berkembang pesat pada saat ini disamping meningkatnya kebutuhan akan komputasi awan berbasis *open source*. OpenStack dapat menangani komputasi dalam skala besar tetapi belum didukung dengan manajemen pemerataan beban kerja perangkat komputasi sehingga dibuat sebuah sistem yang dapat mengatur pemerataan beban kerja komputasi pada komputasi awan secara *real time*.

Pemerataan beban kerja komputasi dilakukan dengan melakukan perpindahan atau *live-migration* mesin virtual dari perangkat komputasi yang memiliki beban kerja berlebih ke perangkat komputasi lain yang memiliki beban kerja komputasi

yang rendah. Pemilihan perpindahan mesin virtual berdasarkan nilai *Minimum Migration Time* paling rendah dan penggunaan CPU paling tinggi pada mesin virtual. Perpindahan mesin virtual dimaksudkan untuk mengurangi beban kerja komputasi dengan membagi beban kerja komputasi ke perangkat yang memiliki beban kerja komputasi yang lebih rendah.

Kemudian sistem diuji fungsionalitas dan performanya. Pengujian dilakukan melalui beberapa skenario yang telah ditentukan. Hasil pengujian menunjukan sistem dapat berjalan dengan baik. Sistem memiliki tingkat efisiensi pembagian beban kerja komputasi yang baik dalam melakukan pemerataan kinerja komputasi.

Kata kunci: OpenStack; Minimum Migration Time; live-migration mesin virtual.

RESOURCE MANAGEMENT FOR IMPROVING CLOUD COMPUTING WORKLOAD EFFICIENCY USING OPENSTACK FRAMEWORK

Student's Name	**:**	**Aji Setyo Utomo**
Student's ID	**:**	**5110 100 010**
Departement	**:**	**Informatics Engineering ,FTIF-ITS**
First Advisor	**:**	**Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**
Second Advisor	**:**	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**

ABSTRACT

Cloud computing is a computing model in resources such as processors, storages, and networks which is provided as a service over the internet or intranet. OpenStack is one of the open-source cloud platform which is focusing on Infrastucure oriented as a Service (IaaS). OpenStack is becoming very popular and growing rapidly accompanied by increased demand for cloud computing based on open source. OpenStack is able to handle large-scale computing, however equalization management workload mechanism for computing device is not yet supported on OpenStack. Therefore it is created a system that can adjust the equalization computing workloads in real time.

Equalization mechanism for computational workload is done by performing live migration of virtual machines from computing device that has excessive workloads to another computing devices that have low computational workload. Selection of virtual machine migration based on the value of the lowest Minimum Time Migration and highest CPU usage on the virtual machine. The purpose of virtual machine migration is to reduce the excessive computing workload by dividing the

computing workloads to computing device that have lower computational workload.

Functionality and performance of the system were tested by performing some predefined scenarios. The results show the system work properly. The system has a high level of efficiency in the distribution of computational workload which is good at doing equalization of computing workload.

Keywords: OpenStack; Minimum Migration Time; live migration virtual machine..

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabil'alamin, segala puji hanya milik Allah SubhanahuWata'alla, atas segala rahmat dan karunia-Nya yang tak terhingga sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "**Manajemen Sumber Daya Untuk Meningkatkan Efisensi Beban Kerja Komputasi Awan Menggunakan Kerangka Kerja Openstack"** dengan baik dan tepat waktu.

Tugas Akhir ini dikerjakan demi memenuhi salah satu syarat guna memperoleh gelar Sarjana Komputer di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya. Penulis menyadari bahwa Tugas Akhir ini bukanlah tujuan akhir dari belajar karena belajar adalah sesuatu yang tidak terbatas.

Penulis mendapatkan banyak sekali doa, bantuan dan dukungan dari berbagai pihak dalam menyelesaikan Tugas Akhir ini. Atas berbagai bantuan dan dukungan tersebut, pada kesempatan ini penulis menghaturkan ucapan terima kasih yang sebesar-besarnya kepada:

1. Allah SWT atas limpahan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik dan tepat waktu.
2. Kedua orang tua penulis, Bapak Drs. Yahmin, MM dan Ibu Setyowati yang tak henti-hentinya memberikan dukungan, semangat, kasih sayang, serta selalu memberikan doa yang tiada habisnya yang dipanjatkan untuk penulis.
3. Kakak Diyah Sukowati, Kakak Muhamad Syahrasi, Kakak Diyah Marhawati, dan Kakak Dyon Aji yang telah memberikan semangat dan doa kepada penulis agar cepat selesai.
4. Bapak Waskitho Wibisono, S.Kom., M.Eng., Ph.D. selaku dosen pembimbing 1, yang telah memberikan kepercayaan,

dukungan, bimbingan, nasehat, serta semangat dikala penulis mengalami kesulitan.

5. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD. selaku dosen pembimbing 2, atas bimbingan, arahan, bantuan serta ide untuk menyelesaikan Tugas Akhir ini.
6. Ibu Nanik Suciati, S.Kom., M.Kom., Dr.Eng. selaku ketua jurusan Teknik Informatika ITS, Prof. Ir. Supeno Djanali, M.Sc, Ph.D selaku dosen wali penulis, yang telah memberikan ilmunya kepada penulis.
7. Nia Pratika Santi yang selalu setia menemani, membantu, memberikan dukungan, doa, serta semangat yang tiada henti-hentinya kepada penulis dengan penuh sabar, suka, duka, keceriaan serta kepedulian yang sangat besar terhadap penulis.
8. Keluarga Griya Asri, Dimas Yoga Pratama, Dicky Budi Aldyato, Wildan Ibrahim, Bramantyo Wido, Ryan Dwi Cahyo, Kesya Din Dalmi, Haryo Triwardhono, Sabila lala, dan Wahyu Firiani yang telah memberikan semangat, keceriaan, dan masukan dalam menyelesaikan Tugas Akhir ini.
9. Keluarga NCC, Fajri Rahmat, Grezio, Ngurah Adi Kusuma, dan teman-teman yang lain selaku teman seperjuangan penulis dan teman berbagi ilmu selama pengerjaan Tugas Akhir ini.
10. Seluruh teman Teknik Informatika ITS angkatan 2010, empat tahun bersama kalian sadar atau tidak telah membentuk karakter dan kepribadian penulis. Terima kasih atas rasa kekeluargaan yang telah kalian berikan kepada penulis. Semoga kita semua sukses di dunia dan akhirat. Tetap semangat kawan.
11. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu terselesaikannya Tugas Akhir ini.

Semoga Allah SWT memberkati dan membalas semua kebaikan yang telah dilakukan. Penulis menyadari masih banyak yang dapat dikembangkan pada Tugas Akhir ini. Oleh karena itu, penulis menerima setiap masukan dan kritik yang diberikan. Semoga Tugas Akhir ini dapat memberikan manfaat.

Surabaya, Juli 2014

Aji Setyo Utomo

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1. Latar Belakang

Perkembangan teknologi komputasi mengalami peningkatan yang sangat signifikan dari waktu ke waktu dan menjadi salah satu aspek yang paling penting dalam perkembangan teknologi informasi. Meningkatnya kebutuhan akan penggunaan teknologi komputasi yang didukung dengan sumber daya *server* dengan skalabilitas dan penyimpanan data yang semakin besar mendorong untuk beralih ke teknologi komputasi berbasis komputasi awan. Teknologi komputasi awan meyediakan layanan penyedia processor (*computing power*), penyimpanan (*storage*), jaringan (*network*) yang kemudian layanan tersebut diberikan kepada pengguna komputasi awan [1]. Dewasa ini banyak penyedia layanan awan yang sedang berkembang di antaranya penyedia infrastruktur sebagai layanan awan seperti OpenStack[1], Eucalyptus[2], CloudStack[3], CloudEra[4], dan OpenNebula[5]. Komputasi awan bisa dikatakan sebagai sumber daya untuk komputasi dan penyimpanan data dalam bentuk mesin virtual. Kehadiran komputasi awan dapat membantu ketersediaan server dan penyimpanan dalam membangun aplikasi yang dijalankan secara virtual.

OpenStack merupakan salah satu kerangka kerja *open source* penyedia infrastuktur komputasi awan. Tujuan OpenStack adalah untuk memungkinkan setiap organisasi atau perusahaan membuat dan menyediakan layanan komputasi awan dengan menggunakan perangkat lunak *open source* yang berjalan di atas perangkat keras yang standar. Di dalam OpenStack terdapat beberapa komponen yang memudahkan pengguna dalam melakukan konfigurasi

[1]OpenStack. https://www.openstack.org/
[2] Eucalyptus. https://www.eucalyptus.com/
[3] CloudStack. http://cloudstack.apache.org/
[4] Cloud Era. http://www.cloudera.com/content/cloudera/en/home/
[5] Open Nebula. http://opennebula.org/

infrastruktur awan, diantaranya adalah: OpenStack Compute, OpenStack Storage, dan OpenStack Image Service.

OpenStack memiliki keunggulan dengan skalabilitas yang besar. Diperlukan alokasi sumber daya (*memory*, CPU (*Central Processing Unit*), *storage*) di awal pembuatan sebuah *server* mesin virtual atau *virtual machine* yang disesuaikan dengan kebutuhan. Tetapi alokasi sumber daya seperti itu dirasa kurang maksimal, karena komputasi dari suatu mesin virtual bisa saja berubah-ubah sesuai tugas yang diberikan secara *real time*. Jika banyak mesin virtual disuatu *host* melakukan komputasi yang besar, maka yang terjadi *host* tersebut akan mengalami *overload* atau beban kerja berlebih, sehingga diperlukan penyetaraan beban kerja mesin virtual ke *host* lain dengan beban kerja yang lebih ringan.

Pada Tugas Akhir ini dibuat sebuah sistem pengolahan sumber daya komputasi awan untuk mengatur efisiensi beban kerja saat komputasi *real time*. Hal ini diperlukan agar kinerja dalam pembagian sumber daya komputasi awan menjadi lebih maksimal.

1.2. Rumusan Permasalahan

Rumusan masalah pada Tugas Akhir ini adalah sebagai berikut.

1. Bagaimana membangun komputasi awan dengan kerangka kerja OpenStack menggunkan sumber daya mandiri.
2. Bagaimana mengukur beban kerja tiap *host* yang bertugas sebagai perangkat komputasi.
3. Bagaimana menentukan *Minimum Migration Time* (MMT) mesin virtual dan CPU *time* pada tiap host yang memiliki beban kerja komputasi berlebih.
4. Bagaimana sistem dapat mendapatkan nilai yang ideal terhadap *host* tujuan untuk melakukan *live-migration* mesin virtual.

1.3. Batasan Masalah

Batasan masalah pada Tugas Akhir ini adalah sebagai berikut.

1. Sistem ini didukung kerangka kerja komputasi awan OpenStack.
2. Sistem ini menggunakan beberapa perangkat keras, yaitu:
   - 4 komputer.
   - 1 *switch.*
3. Sistem hanya dapat melakukan proses *live-migration* untuk *host* yang *overload*.
4. Sistem hanya menggunakan satu sistem operasi yang sama di tiap mesin virtual.

1.4. Tujuan dan Manfaat

Tujuan dari pengerjaan Tugas Akhir ini adalah sebagai berikut.

1. Membuat sistem yang dapat melakukan estimasi beban kerja tiap *host* komputasi yang bertugas sebagai perangkat komputasi secara *real time*.
2. Sistem dapat menentukan mesin virtual yang memiliki *Minimum Migration Time* paling rendah dan beban CPU paling tinggi.
3. Sistem dapat melakukan *live-migration* secara dinamis ke *host* lain dengan nilai yang paling ideal .

Manfaat dari pengerjaan Tugas Akhir ini adalah sebagai berikut.

1. Melakukan pembagian beban kerja komputasi pada perangkat komputasi yang mengalami beban kerja berlebih.
2. Melakukan perpindahan mesin virtual dari perangkat komputasi satu ke perangkat komputasi lain secara dinamis, dengan dipengaruhi nilai ketersediaan sumber daya pada suatu perangkat komputasi.

1.5.Metodologi

Adapun langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini adalah sebagai berikut.

1. Penyusunan proposal tugas akhir

   Proposal Tugas Akhir ini berisi tentang sistem manajemen sumber daya komputasi awan yang dapat memaksimalkan penggunaan sumber daya pada pada *host* secara *real time*. Sistem ini dapat mengatur beban komputasi pada setiap *host*, dimana beban kerja tiap perangkat komputasi akan selalu berubah setiap waktu tergantung beban kerja mesin virtual yang dilakukan. Untuk itu diperlukan penempatan ulang mesin virtual dengan proses *live-migration* secara *real time.* Sehingga sistem akan membuat proses komputasi semua *host* menjadi optimal.

2. Studi literatur

   Tahap ini merupakan tahap pengumpulan informasi dan pembelajaran yang akan digunakan pada Tugas Akhir ini. Studi literatur meliputi diskusi dan pemahaman mengenai topik yang berhubungan dengan Tugas Akhir, diantaranya mengenai:

   a. Implementasi OpenStack pada CentOS[6] menggunakan KVM[7] Hypervisor dan GlusterFS[8] Distributed File System.
   b. Refrensi dari *"OpenStack Neat: a framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds"* oleh Anton Beloglazov dan Rajkumar Buyya dari Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Melbourne, Australia [2].
   c. *Minimum Migration Time* untuk perpindahan mesin virtual.

---

[6] CentOS. https://www.centos.org/

[7] KVM. http://www.linux-kvm.org/page/Main_Page/

[8] GLusterFS. http://www.gluster.org/

3. Perancangan sistem

Sistem manajemen sumber daya komputasi awan adalah sistem yang dapat membagi beban kerja komputasi antar perangkat komputasi. Sistem dapat mendapatkan nilai komputasi yang ideal untuk dipindahkan secara *real-time*. Langkah awal adalah mendapatkan nilai dari banyak *host* yang bertugas sebagai perangkat komputasi. Kemudian data disimpan ke dalam satu *database* di perangkat kontroler yang bertugas sebagai *controller*. Langakah selanjutnya adalah sistem akan memeriksa perangkat komputasi yang memiliki beban kerja komputasi berlebih.

Dari data tersebut, sistem akan mencari mesin virtual dengan *Minimum Migration Time* (MMT) paling minimum. Setelah itu sistem akan memeriksa perangkat komputasi dengan beban kerja rendah yang memiliki ketersediaan sumber daya ketika mesin virtual dipindahkan. Jika ternyata sumber daya dari semua perangkat komputasi tidak tersedia, maka sistem akan mencari mesin virtual dengan status tidak aktif atau *suspend*. Mesin virtual yang memiliki status tidak aktif akan dialokasikan untuk dipindahkan ke perangkat lain lain agar mesin virtual yang aktif di perangkat komputasi yang memiliki beban kerja berlebih dapat menggantikan mesin virtual yang tidak aktif. Sistem akan melakukan perpindahan mesin virtual secara *real time* tergantung beban kerja yang diberikan ke perangkat komputasi, sehingga kinerja semua host dapat optimal.

4. Implementasi perangkat lunak

Tahap ini merupakan implementasi rancangan sistem yang telah dibuat. Tahap ini merealisasikan apa yang terdapat pada tahapan perancangan yang telah dibuat sebelumnya sehingga menjadi sebuah rancang bangun sistem yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi

Dalam tahap ini pengujian akan dilakukan pada empat komputer dengan satu komputer bertindak sebagai kontroler dan

tiga komputer lain sebagai komputasi. Dimana pengujian akan dilakukan dengan memberikan beban kerja yang berbeda pada tiap perangkat komputasi dan tiap mesin virtual sehingga nantinya akan terlihat proses perpindahan mesin virtual dari satu perangkat ke perangkat komputasi lain sehingga dapat membuat beban kerja tiap host menjadi optimal. Tahap evaluasi akan dilakukan berdasarkan hasil dari tahap pengujian.

6. Penyusunan buku tugas akhir.

Pada tahap ini disusun laporan tugas akhir sebagai dokumentasi pelaksanaan tugas akhir, yang mencakup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.6. Sistematika Laporan

Buku Tugas Akhir ini disusun dengan sistematika laporan sebagai berikut.

1. **Bab I. Pendahuluan**
   Bab ini meliputi latar belakang masalah, rumusan permasalahan, batasan masalah, tujuan dan manfaat pembuatan Tugas Akhir, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.
2. **Bab II. Tinjauan Pustaka**
   Bab ini meliputi dasar teori dan penunjang yang berkaitan dengan pokok pembahasan dan mendasari pembuatan Tugas Akhir ini.
3. **Bab III. Perancangan Perangkat Lunak**
   Bab ini membahas desain dari sistem yang akan dibuat meliputi arsitektur sistem, dan diagram aliran data sistem.
4. **Bab IV. Implementasi Perangkat Lunak**
   Bab ini membahas implementasi dari desain sistem yang dilakukan pada tahap desain, meliputi potongan

pseudocode dan implementasi antarmuka dari perangkat lunak.

5. **Bab V. Uji Coba dan Evaluasi**
   Bab ini membahas uji coba dari perangkat lunak yang dibuat dengan melihat keluaran yang dihasilkan oleh perangkat lunak, analisa, dan evaluasi untuk mengetahui kemampuan perangkat lunak.

6. **Bab VI. Kesimpulan dan Saran**
   Bab ini berisi kesimpulan dari hasil uji coba yang dilakukan serta saran untuk pengembangan lebih lanjut perangkat lunak.

BAB II
TINJAUAN PUSTAKA

Bab ini berisikan mengenai teori-teori yang berkaitan dengan implementasi perangkat lunak, dengan tujuan memberikan gambaran secara umum mengenai sistem yang dibangun dan berguna dalam pengembangan Tugas Akhir ini.

2.1. Komputasi Awan

Komputasi awan merupakan sebuah model komputasi pada sumber daya seperti prosesor (*computing power*), penyimpanan (*storage*), jaringan (*network*) menjadi abstrak dan diberikan sebagai layanan di jaringan atau internet. Ketersediaan *on-demand* sesuai kebutuhkan, mudah untuk melakukan pengaturan, dinamik dan skalabilitas yang hampir tanpa limit adalah beberapa atribut penting dari komputasi awan [3]. Terdapat tiga layanan yang disediakan oleh komputasi awan diantaranya sebagai berikut.

1. *Software as a Service* (SaaS)
   SaaS adalah layanan dari komputasi awan dimana *pengguna* dapat menggunakan perangkat lunak yang telah disediakan oleh penyedia layanan awan.
2. *Platform as a Service* (PaaS)
   Paas adalah layanan dari komputasi awan dimana pengguna dapat mengembangkan komputasi awan menjadi layanan SaaS. Biasanya sudah terdapat sistem operasi, *database*, *web server* dan kerangka kerja aplikasi agar dapat menjalankan aplikasi yang telah dibuat. Keuntungan layanan PaaS ini bagi pengembang adalah mereka dapat fokus pada aplikasi yang mereka buat tanpa memikirkan pemeliharaan dari *computing platform*.
3. *Infrastructure as a Service* (IaaS)
   IaaS adalah layanan dasar dari komputasi awan, di layanan ini pengguna dapat merancang dan membangun sistem komputasi

awan sendiri. Sebagai contoh, saat komputer virtual tersebut sudah kelebihan beban, kita bisa tambahkan CPU, *RAM*, dan *storage*.

Dalam Tugas Akhir ini difokuskan dalam *low level service model* yaitu implementasi komputasi awan dalam IaaS. Ada tiga dasar pembuatan komputasi awan diantaranya sebagai berikut.

1. *Public Cloud*: infrastuktur dari komputasi awan yang didukung layanan internet untuk layanan ke publik.
2. *Private Cloud*: infastuktur komputasi awan yang dapat dibuat di suatu organisasi, dan operasinya bersifat internal.
3. *Hybrid Cloud*: infrastuktur komputasi awan yang mengabungkan *private* dan *public cloud*.

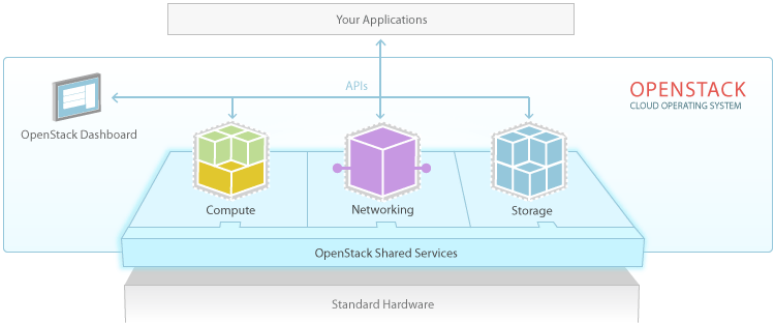
Public Clouds, seperti Amazon EC2[9] merupakan awal dari industri komputasi awan. Namun, penggunaan platform tersebut masih *closed source* sehingga pengembang mengalami kesulitan ketika membangun perangkat lunak diatas komputasi awan. Kemudian muncul beberpa komputasi awan yang *open source* seperti OpenStack[1], Eucalyptus[2], OpenNebula[5] dan CloudStack[3]. Beberapa *paltform* tersebut dapat digunakan tidak hanya membangun di lingkungan *private cloud*, tetapi juga dapat berkontribusi untuk membangun *paltfrom* tersebut.

2.2. OpenStack

OpenStack merupakan software *open-source* dalam komputasi awan yang berorientaasi di bidang *Infrastructure as Service* (IaaS). OpenStack pertama kali dirilis oleh Rackspace dan NASA dibawah lisensi Apache 2.0 pada July 2010. OpenStack mengatur semua proses komputasi dan sumber daya baik jaringan

[9] Amazon EC2. http://aws.amazon.com/ec2/

ataupun sumber daya perangkat keras melalui *dashboard* yang memberikan control administrasi sekaligus hak akses pada pengguna melalu web antarmuka [4]. OpenStack juga menyediakan API yang kompatibel dengan Amazon EC2 dan mendukung Open Cloud Computing Interface (OCCI)[10].



Gambar 2.1 Gambaran umum arsitektur OpenStack [5]

OpenStack dibagi menjadi beberapa layanan yang ditunjukan pada Gambar 2.1. Layanan yang ada pada OpenStack diantaranya adalah sebagai berikut.

1. *OpenStack Compute (Nova)*: mengatur kinerja mesin virtual dari sumber daya yang tersedia baik untuk *migration* dan jaringan. Untuk penggunaan API virtualisasi menggunkan Libvirt. *OpenStack Compute* mendukung banyak hypervisors, seperti KVM[7] dan Xen[11].
2. *OpenStack Storage*: mendukung *block* dan *object storage* yang digunakan mesin virtual. *Block storage* digunakan untuk membuat sistem *block storage* dari mesin virtal atau *virtual machine* (VM) dengan menggunakan *dashboard* atau API. Pada penambahan *block storage* terdapat distiribusi yang dapat disesuaikan sesuai kebutuhan *object storage* yang dapat diakses menggunakan API.

[10] OCCI http://occi-wg.org/

[11] XEN. http://www.xenproject.org/developers/teams/hypervisor.html

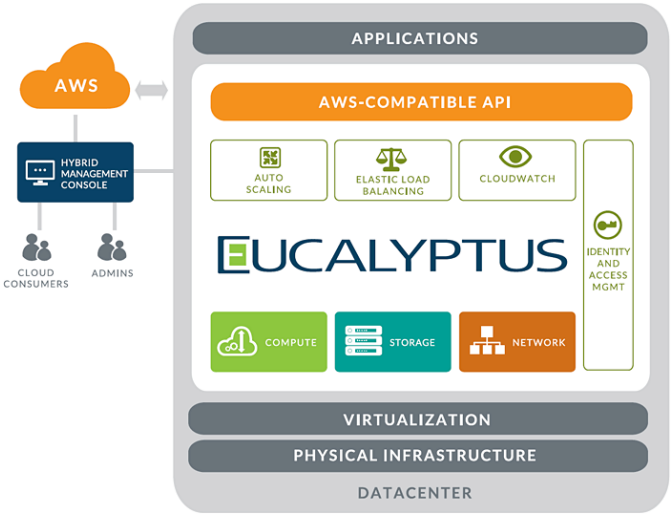
3. *OpenStack Networking*: mendukung API dari network dan manajemen alamat IP. Sistem akan mengijinkan pengguna untuk membuat sendiri jaringan dan mengatur alamat IP untuk VM.
4. *OpenStack Dashboard (Horizon)*: sebuah halaman web antar muka untuk memudahkan admin dan pengguna untuk mengatur semua kinerja dari VM, manajemen VM *image,* dan manajemen *storage*.
5. *OpenStack Identity (Keystone)*: merupakan sebuah pusat manajemen layanan akun pengguna untuk autentikasi dan akses pada control sistem. Disamping itu servise digunakan untuk ases ke dalam daftar layanan OpenStack yang meliputi *data center* dan kominikasi *endpoints*.
6. *OpenStack Image (Glance):* merupakan manajemen berbagai macam VM *image* diantaranya Raw, AMI, VHD, VDI, qcow2, VMDK, dan OVF. *Openstack Image* mengatur kinerja *image* mulai dari registrasi, penyaluran *image* ke VM dan *snapshoting*.

Dalam membangun OpenStack dengan banyak *host* komputasi diperlukan beberepa faktor pendukung sehingga OpenStack dapat bekerja secara maksimal. Dalam Tugas Akhir ini menggunkan beberapa perangkat pendukung seperti GlusterFs sebagai perangkat lunak untuk mendukung distribusi *file system* dan KVM yang merupakan faktor pendukung dalam proses virtualisasi untuk Linux.

2.2.1. Perbandingan *Open Source Cloud Platforms* lain dengan OpenStack

OpenStack merupakan salah satu dari banyak *Open Source Cloud Platforms* yang menyediakan fasilitas untuk komputasi awan. Secara garis besar perbedaan Openstack dengan tiga *Open Source Cloud Platforms* yang lain seperti Eucalyptus, Open Nebula, dan CloudStack.

Eucalyptus adalah salah satu *Open Source Cloud Platforms* yang dibuat oleh Eucalypus Systems pada bulan Maret 2008 dibawah lisensi GPL v3. Eucalyptus yang memiliki kepanjangan "Elastic Utility Computing Architecture for Linking Your Programs to Useful Systems" [6]. Pada awal pembuatan Eucalyptus versi 3.1 dibuat dua edisi yaitu *open source* dan *enterprise* yang dimana dalam versi *enterprise* banyak fitur yang ditambahkan. Kemudian di versi berikutnya dua edisi dari versi 3.1 digabungkan menjadi satu proyek *open-source.* Arsitektur Eucalyptus dapat dilihat pada Gambar 2.2.



Gambar 2.2 Arsitektur Eucalyptus [7]

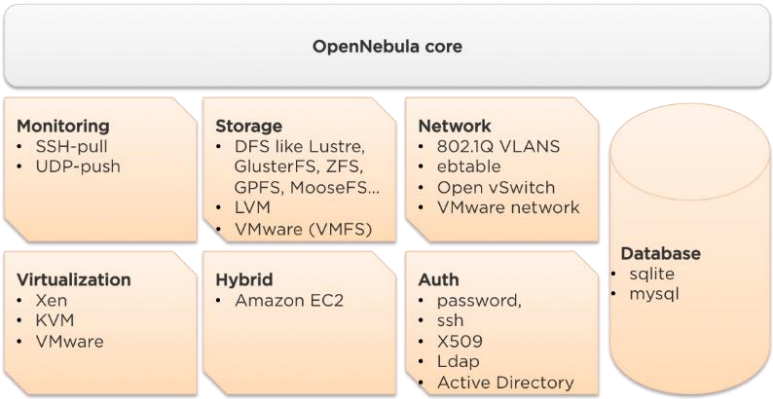
Pada bulan Maret 2012, Eucalyptus dan Amazon Web Services (AWS) bekerjasama dan membuat API yang saling mendukung, diantaranya menambahkan fitur *workload migration* dan *hybrid Cloud enviroments* [6]. Eucalyptus terdiri dari 5 componen *high-level* diantaranya sebagai berikut.

1. *Cloud Controller*: mengatur sumber daya pokok virtualisasi (*servers*, *network*, dan *storage*) yang didukung dengan antar muka web dan API yang kompatibel dengan Amazon EC2
2. *Cluster Controller*: mengatur VM yang ada dibeberapa titik komputasi dan mengatur jaringan antar VM, dan VM dengan pengguna.
3. *Walrus:* implementasi *object storage* yang mengakses API yang kompatibel dengan Amazon S3
4. *Storage Controller*: mendukung untuk penempatan *block storage* secara dinamik pada VM dengan API Amazon Elastic Block Storage (EBS).
5. *Node Controller*: mengatur siklus dari VM yang berjalan di atas *physical node* menggunkan fungsi yang didukung oleh *hypervisor.*

OpenNebula merupakan *open source* IaaS *Cloud paltorm* yang didirikan sebagai sebuah proyek penelitian di tahun 2005 oleh Ignacio M. Llorente dan Ruben S. Montero. Perangkat lunak ini pertama kali dipublikasikan pada bulan Maret 2008 dibawah lisensi Apache 2.0. Pada Maret 2010 pendiri dari OpenNebula membuat C12G Labs, sebuah organisasi yang bertujuan mendukung ke arah komersial dan menambah servis untuk perangkat lunak OpenNebula [8]. Saat ini proyek OpenNebula diatur oleh C12G Labs. OpenNebula mendukung beberapa API seperti EC2 query, OGF OCCI, dan VCloud. Arsitektur OpenNebula dapat dilihat pada Gambar 2.3. OpenNebula mendukung beberapa fitur dan komponen. *User dan Groups*: OpenNebula mendukung banyak akun pengguna dan banyak kelompok dengan berbagai pengenalan sesuai hak akses dari managemen atau disebut Acces Control List (ACL).

1. *Virtualization Subsystem*: mengatur komunikasi dengan *hypervisor* yang dipasang pada *physical host* untuk menghubungkan manajemen dan pengawasan dari kinerja banyak VM.

2. *Network Subsystem*: mengatur jaringan virtual yang mendukung VM yang terhubung dengan VLANs dan Open vSwitch.
3. *Storage Subsystem*: mendukung beberapa jenis data penyimpanan untuk VM *images*.
4. *Clusters*: merupakan bagian terpenting utnuk menghubungkan *data stores* dan *virtual network* yang dapat digunkan untuk proses *load balancing*, *high availability*, dan peningkatan performa komputasi.



Gambar 2.3 Arsitektur Open Nebula [9]

CloudStack merupakan *open source* IaaS *Cloud paltorm* yang awalnya dibuat oleh Cloud.com. Pada Mei 2010, kebanyakan dari perangkat lunak dibuat dibawah GPL v3 license, hampir 5% dari sumber kode tidak dibagikan untuk umum. Pada Juli 2011, Citrix membeli Cloud.com dan pada April 2012, Citrix donasikan CloudStack kepada Apache Software Fondation dan mengganti ke license Apache 2.0. Dalam CloudStack terdapat Amazon EC2 dan S3 APIs, seperti vCloud API, dan menambahkan beberapa API sendiri [10]. CloudStack mempunyai struktur yang bertingakat, seperti manajemen dari banyak *physical host* dari satu halaman muka. Strukur yang ada di CloudStack diantaranya:

1. *Availability Zones*: merupakan diskripsi dari lokasi yang digunakan untuk alokasi VM di *data storage*. Sebuah *Availability Zones:* terdiri dari Pod dan *Secondary Storage*.
2. *Pods*: pengaturan perangkat kelas dari *Clusters*. Sebuah Pod dapat terdiri dari satu atau lebih *Clusters*, dan sebuah *Layer* 2 *switch* arsitektur.
3. *Clusters*: merupakan kelompok-kelompok dari identifikasi *physical host* yang sedang berjalan di *hypervisor* yang sama. Sebuah *Cluster* mempunyai *dedicated primary storage device*, didamana VM terdapat di dalam *host*.
4. *Primary Storage*: merupakan salah satu bagian dari *Cluster* yang digunakan untuk *storage* VM.
5. *Secondary Storage*: digunakan untuk menyimpan VM *images* dan *snapshots*.

Dari perbandingan ketiga *cloud platforms* dapat disimpulakan pada Tabel 2.1.

Tabel 2.1. Perbandingan OpenStack, Eucalyptus, OpenNebula, dan CloudStack [11]

	OpenStack	**Eucalyptus**	**OpenNebula**	**CloudStack**
Dikelola	OpenStack Foundation	Eucalyptus Systems	C12G Labs	Apache Software Foundation
Lisensi	Apache 2.0	GPL v3	Apacahe 2.0	Apache 2.0
Rilis	Oktober 2010	Mei 2010	Maret 2008	May 2010
Kompatibel OCCI	Yes	No	Yes	No
Kompatibel AWS	Yes	Yes	Yes	Yes
Hypervisors	Xen, KVM, VMware	Xen, KVM, VMware	Xen, KVM, VMware	Xen, KVM, VMware, Oracle VM
Programming Language	Python	Java, C	C, C++, Ruby, java	Java

2.3. CentOS

CentOS Linux atau Community Enterprise Operating System merupakan salah sistem operasi Linux yang secara bebas diberikan kepada publik dari Red Hat untuk Red Hat Enterprise Linux (RHEL) [12]. CentOS Linux bertujuan untuk menggabungkan fungsional dengan RHEL. Distribusi CentOS linux dilakukan bebas dan tanpa biaya. Setiap versi CentOS dipertahankan hingga 10 tahun (dengan pembaruan keamanan dengan durasi dukungan dari Red Hat yang bervariasi tergangung sumber yang dirilis). Versi CentOS dirilis 2 tahun dan setiap versi CentOS secara berkala diperbarui sekitar 6 bulan untuk mendukung perangkat keras baru. Centos memiliki keamanan yang aman, pemeliharaan yang mudah, dapat diprediksi dan diproduksi oleh linux.

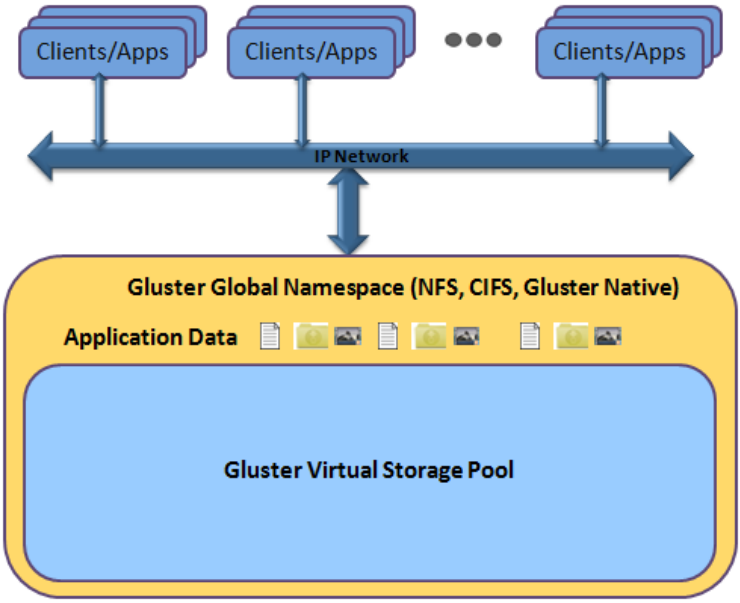
Dalam Tugas Akhir ini menggunkan sistem operasi CentOS sebagai penyedia sistem operasi dimana menjadi dasar dari perangkat lain untuk diimplementasikan.

2.4.GlusterFS

GlusterFS merupakan perangkat lunak open source yang mendukung distribusi *file system* yang mampu mengatur skalabilitas dengan beberapa *petabyte* dan mampu mengatur banyak klien. *Cluste*r GlusterFS dapat membangun blok diatas TCP atau IP yang saling terhubung. GlusterFS juga menggabungkan sumber daya *hard drive* dan *memory* yang kemudian data akan dikelola kedalam sebuah *global namespace* [13].

GlusterFs mendukung klien yang menjalankan aplikasi diatas jaringan standar IP. Gambar 2.1 merupakan ilustarasi bagaimana pengguna dapat mengakses aplikasi data dan file di *Global Namespace* dengan menggunakan standar protokol.

GlusterFs memudahkan pengguna dalam membangun virtualisasi *storage* baik dari *petabytes* ataupun *terabytes*.



Gambar 2.4 Gambaran umum GlusterFs [13]

Dalam Tugas Akhir ini menggunakan GlusterFs untuk mendukung distribusi *file system* dan membagi *storage system* antara perangkat komputasi. GlusterFS juga mendukung untuk proses *live migration* dari mesin virtual. Penggunaan layanan *storage* yang terpusat akan sangat terbatas dan alokasi *storage* mempunyai resiko cenderung gagal. Untuk itu diperlukan distribusi *file system* seperti GlusterFs. Dengan penggunaan GlusterFs memiliki beberapa keuntungan diantaranya.

- Tidak ada *single point failure*, dikarenakan *storage* dan data di distribusikan ke server lain secara otomatis sehingga jika salah satu server gagal maka server yang lain dapat menampung.
- *Higher scalability*, masukan dan pengeluaran (I/O) dapat berjalan di banyak server.

2.5.QEMU

QEMU adalah sembuah dasar dan *open source* untuk virtualisasi dan emulator. Ketika menggunakan mesin emulator, QEMU dapat menjalankan sistem operasi dan program di suatu mesin (virtual) yang berjalan diatas mesin (PC). QEMU akan mendukung virtualisasi jika dijalankan dengan *hypervisor* lain seperti KVM [14].

2.6.Libvirt

Libvirt adalah sembuah perangkat lunak yang menyediakan cara mudah untuk mengelola mesin virtual dengan banyak fungsionalitas virtualisasi lainya, seperti penyimpanan dan manajemen jaringan. Bagian yang terdapat dari Libvirt termasuk API *library,* daemon (Libvirtd), dan *command line utility* (Virsh) [15]. Tujuan utama dari Libvirt adalah sebagai penyedia fasilitas pengelola banyak virualisasi yang berbeda dalam *hypervisor*. Dalam Tugas Akhir ini Libivirt digunakan untuk mendukung perangkat *hypervisor* untuk mengelola virtualisasi yang ada pada perangkat komputasi.

2.7.KVM

KVM atau Kernel-based Virtual Machine adalah salah satu solusi virtualisasi yang berbasis *open source* dari Linux yang mendukung beberpa ekstensi misalnya Intel VT atau AMD-V [16]. KVM juga mendukung virtualisasi sesuai *core* yang tersedia di perangkat keras. KVM bertugas sebagai *hypervisor* yang didukung oleh QEMU untuk membuat mesin virtual dan menjalankan banyak mesin virtual sekaligus. Mesin virtual yang telah dibangun dalam KVM akan mempunyai *network card, storage, memory* dan *graphics adapter* yang berjalan diatas *physical machine.* Dalam Tugas Akhir ini menggunakan KVM sebagai perangkat lunak yang mendukung proses virtualisasi pada pembuatan mesin virtual.

2.8. Python

Python merupakan bahasa pemrograman yang multiguna dengan perancangan yang berfokus pada tingkat keterbacaan kode [17]. Python diklaim sebgai bahasa pemrograman yang terdiri dari sintaksis kode yang jelas dan dilengkapi fungsionalitas pustaka standar yang saling mendukung. Salah satu fitur yang tersedia di python adalah sebagai pemrograman dinamis yang dilengkapi dengan manajemen memori yang otomatis. Python biasanya digunakan untuk keperluan pengembangan perangkat lunak. Saat ini python dapat dijalankan di berbagai sistem operasi. Dalam Tugas Akhir ini menggunakan bahasa pemrograman python untuk mengatur sistem yang dinamik dalam perpindahan mesin virtual.

2.9.MySQL

MySQL merupakan salah satu sistem manajemen basis data open source yang dibangun, didistribusikan, dan didukung oleh Oracle Corporation [18]. Pada MySQL terdapat *logical model* yang terdiri dari *databases*, *tables*, *views*, *row*, dan *columns* untuk membantu penggunaan data yang terstuktur di lingkungan programer. MySQL juga mendukung sitem klien server, sehigga data dapat terpusat di server dan dapat diakses oleh banyak klien. Dalam Tugas Akhir ini menggunakan MySQL untuk manajemen data yang didapat dari semua proses komputasi awan, baik dari perangkat komputasi dan maupun *virtual machine*.

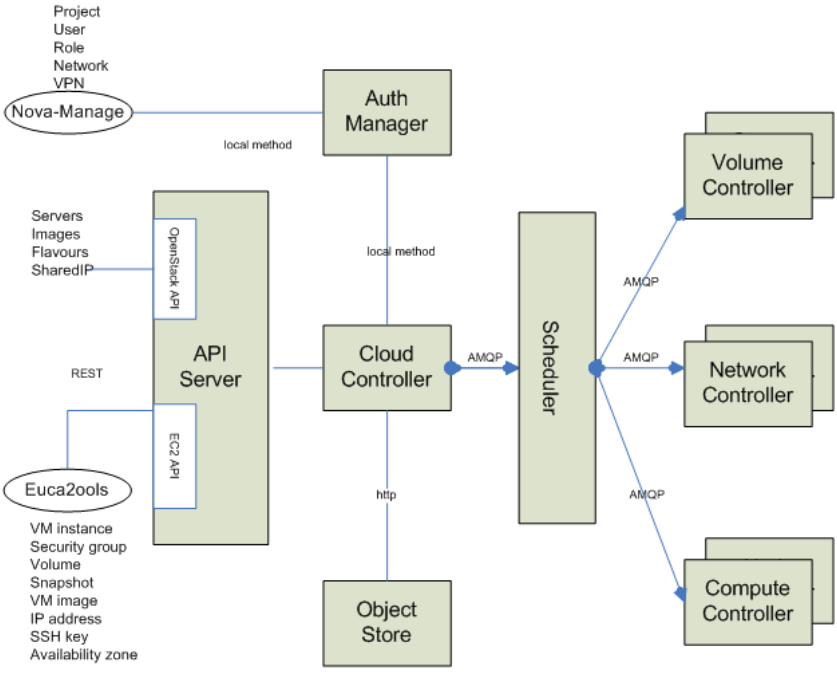
2.10. CirrOS

CirrOS adalah minimal Linux instalasi yang didesain minil untuk pengujian komputasi awan seperti OpenStack-Compute [21]. Dalam Tugas Akhir ini menggunakan OS CirrOS sebagai sistem operasi dasar pada mesin virtual.

2.11. Qpid AMQP Message Broker

Qpid AMQP message broker adalah platform yang digunakan menyampaikan pesan AMQP dan mendukung penyedia pesan untuk banyak platform bahasa pemrograman [19]. Teknologi AMQP mendukung perangkan komputasi awan OpenStack. AMQP merupakan penyedia pesan disamping RabitMQ yang berada pada komponen nova untuk saling terhubung. Pada openstack penggunaan *broker* dapat dijelaskan seperti Gambar 2.5.Fitur dari AMQP diantaranya sebagai berikut.

- Menghubungkan klien dan server.
- Sinkronasi penuh diantara klien dan penyedia server.
- Mengatur keseimbangan perangkat



Gambar 2.5 Gambaran pengunaan AMQP [20]

BAB III
PERANCANGAN PERANGKAT LUNAK

Pada bab ini dijelaskan mengenai dasar perancangan perangkat lunak yang dibuat dalam Tugas Akhir ini. Secara khusus akan dibahas mengenai deskripsi umum sistem, perancangan proses, alur, serta gambaran implementasi sistem yang diterapkan pada komputasi awan berbasis OpenStack.

3.1. Diskripsi Umum Sistem

Pada Tugas Akhir ini dibangun sistem manajemen sumber daya mesin virtual yang dapat mengatur perpindahan mesin virtual secara dinamik dan *real time* tergantung beban kerja komputasi yang diberikan kepada perangkat (*host*) komputasi. Sistem akan mencatat data dari semua perangkat komputasi mulai dari beban kerja perangkat komputasi, RAM, *storage*, dan juga mesin virtual yang berjalan di atas perangkat komputasi. Data yang disimpan dapat berubah secara dinamik sesuai beban kerja mesin virtual yang diberikan kepada perangkat komputasi. Dari data tersebut, sistem akan menentukan perangkat komputasi mana yang mengalami beban kerja komputasi berlebih atau *overload*, sehingga sistem akan menentukan perpindahan mesin virtual agar beban kerja semua perangkat komputasi dapat optimal.

Sistem yang dibangun ini berjalan diatas komputasi awan dengan *framework* OpenStack dan dibangun dengan sumber daya mandiri. Sistem terdiri dari satu perangkat sebagai kontroler dan tiga perangkat lain sebagai *host* komputasi. Kontroler bertugas untuk mengatur semua aktifitas di semua mesin virtual. Sedangkan pada perangkat komputasi bertugas sebagai penyedia sumber daya untuk mesin virtual. Jumlah dari perangkat komputasi dapat berubah sesuai kebutuhan komputasi. Sistem akan berjalan diatas semua perangkat komputasi dan perangkat kontroler yang saling terhubung. Penyimpanan semua data dari proses komputasi akan terpusat di *database* yang ada di perangkat kontroler. Hal ini

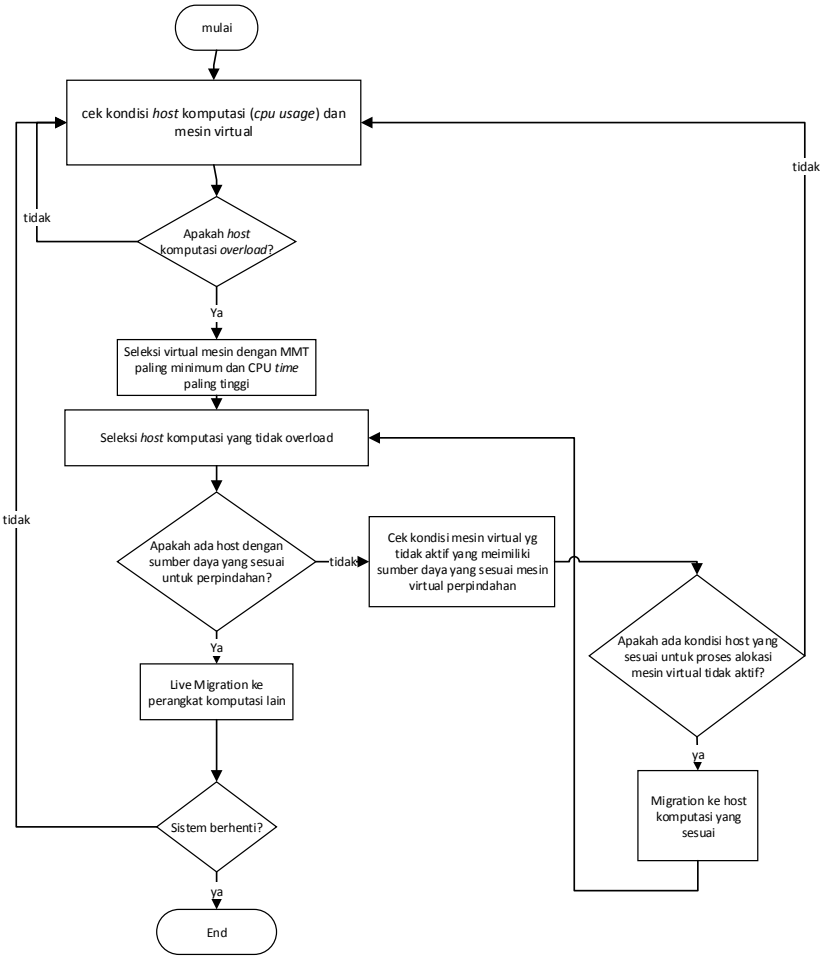
bertujuan untuk mempermudah mengola data yang ada di semua perangkat komputasi.

Pada perangkat komputasi, sistem akan mencatat data beban kerja CPU, kondisi ketersediaan RAM, ketersediaan penyimpanan dan juga akan mencatat semua aktifitas mesin virtual yang ada di perangkat komputasi secara *real-time.* Hal ini bertujuan untuk memantau kondisi di suatu perangkat komputasi.

Pada perangkat kontroler, sistem secara *real-time* juga memeriksa data yang telah dikirim oleh semua perangkat komputasi untuk menentukan perangkat komputasi yang mengalami kelebihan kinerja. Jika terdapat perangkat komputasi yang mengalami kelebihan beban kerja, maka sistem akan memeriksa data dari mesin virual yang terdapat di perangkat komputasi dan mencari *Minimum Migration Time* (MMT) paling kecil dari mesin virtual. *Minimum Migration Time* (MMT) adalah waktu yang digunakan perangkaat kumputasi untuk memindahkan mesin virtual ke perangkat komputasi lain. Sistem juga akan mencari mesin virtual yang paling banyak melakukan pekerjaan, sebab akan lebih efisien dalam pengurangan beban kerja perangkat komputasi yang mengalami kelebihan beban. Perpindah mesin virtual ke perangkat komputasi lain dapat dilakukan jika alokasi sumber daya perangkat komputasi tujuan sesuai dengan mesin virtual yang akan dipindah. Sistem yang berjalan di perangkat kontroler juga bertugas untuk melakukan perpindahan mesin virtual sesuai parameter yang menunjukan perpindahan paling optimal.

3.2. Arsitektur Umum Sistem

Agar dapat menjalankan fungsinya, maka alur kerja dari kesatuan sistem ini dirancang seperti pada Gambar 3.1.

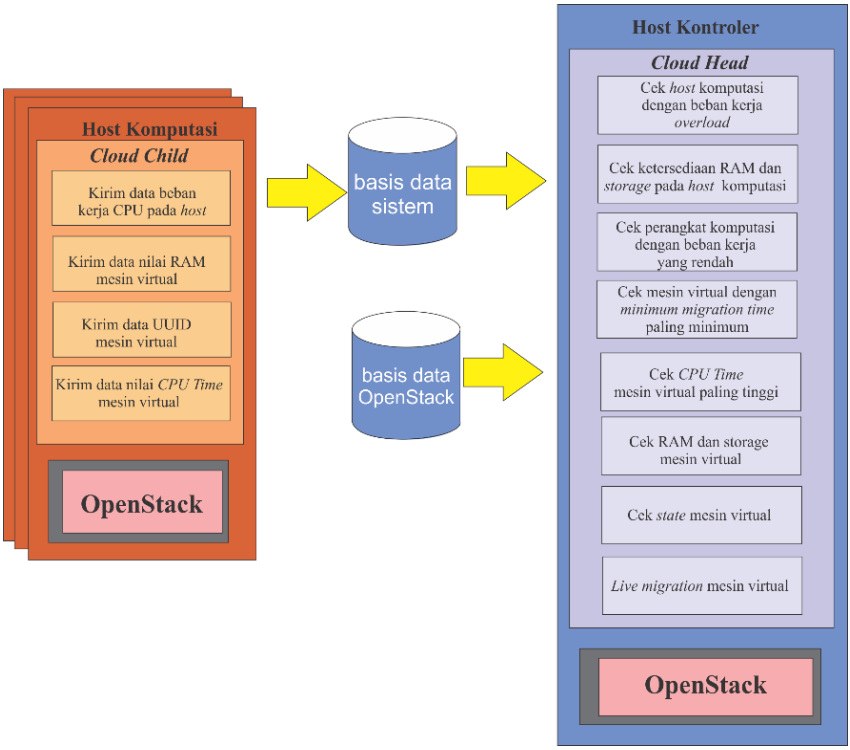


Gambar 3.1 Gambaran Alur Kerja Sistem

Berdasarkan Gambar 3.1, alur kerja sistem ini dijabarkan sebagai berikut.

1. Sistem akan memeriksa beban kerja perangkat komputasi dan kondisi semua mesin virtual yang kemudain data tersebut akan disimpan ke dalam *database.*
2. Sistem akan memeriksa data dan menentukan perangkat komputasi yang mengalami beban kerja berlebih.
3. Sistem akan mencari mesin virtual yang mempnyai *Minimum Migration Time* (MMT) paling minimum dan mesin virtual yang melakukan beban kerja yang paling berat dari perangkat komputasi yang mengalamai beban kerja berlebih.
4. Dari mesin virtual yang mempunyai MMT paling minimum sistem akan menentukan perangkat komputasi lain sebagai tujuan dengan ketentuan bahwa alokasi RAM perangkat komputasi tersedia untuk mesin virtual yag akan di pindah.
5. Jika alokasi RAM dan *storage* dari perangkat tujuan tersedia, makan *live migration* dapat dilaksanakan dengan tujuan perangkat komputasi yang lebih ideal.
6. Jika alokasi RAM tidak tersedia di semua perangkat komputasi, sistem akan mencari perangkat komputasi yang memiliki mesin virtual dengan status tidak aktif pada perangkat komputasi yang tidak mengalami beban kerja berlebih.
7. Dari mesin virtual yang memiliki status tidak aktif, sistem akan mengalokasikan perangkat komputasi lain untuk dipindahkan.
8. Jika tetap tidak ditemukan perangkat komputasi lain yang sesuai untuk perpindahan host yang tidak aktif, maka sistem tidak dapat melakukan *live migration* untuk perangkat yang memiliki beban kerja berlebih.
9. Jika ditemukan perangkat komputasi yang sesuai untuk mesin virtual yang aktif, maka sistem akan mengalokasikan mesin virtual yang tidak aktif untuk perangkat komputasi yang memiliki beban kerja berlebih.
10. Sistem dapat melakukan *live migration* dengan tujuan perangkat komputasi baru.

Pada Gambar 3.2 merupakan gambaran arsitektur dari sistem manajemen sumber daya yang terdiri dari dua modul. Modul pertama adalah *Cloud Child* yang merupakan bagian dari sistem yang terdapat di semua perangkat komputasi, dimana semua aktifitas pengecekan kondisi perangkat komputasi dilakukan di modul ini. Pengecak kondisi semua perangkat disimpan kedalam satu *database* terpusat yang ada di perangkat kontroler.



Gambar 3.2 Arsitektur Sistem Manajemen Sumber Daya

Modul kedua sistem adalah C*loud Head* merupakan bagian sistem yang terdapat pada perangkat kontroler, bagian ini bertugas untuk mengolah data yang disimpan dari perangkat komputasi. Dari data perangkat komputasi sistem akan melakukan beberapa

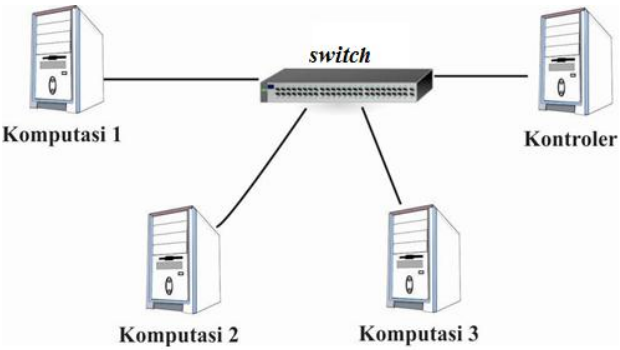
pengecekan yang kemudian didapatkan data perpindahan mesin virtual yang paling ideal untuk perangkat kumputasi yang memiliki beban kerja berlebih. Setelah didapat data perpindahan sistem akan melakukan perpindahan mesin virtual ke perangkat komputasi lain dengan proses *live migration.*

3.3. Perancangan Komputasi Awan

Pada sistem ini menggunakan empat komputer untuk membangun jaringan komputasi awan OpenStack. Satu komputer sebagai kontoler, tiga lain sebagai perangkat komputasi. Semua perangkat saling terhubung dengan dengan sebuah switch.

Perangkat kontroler bertugas untuk melayani *service* dari OpenStack. Perangkat kontroler juga bertugas sebagai *server* dan manajemen semua perangkat komputasi.

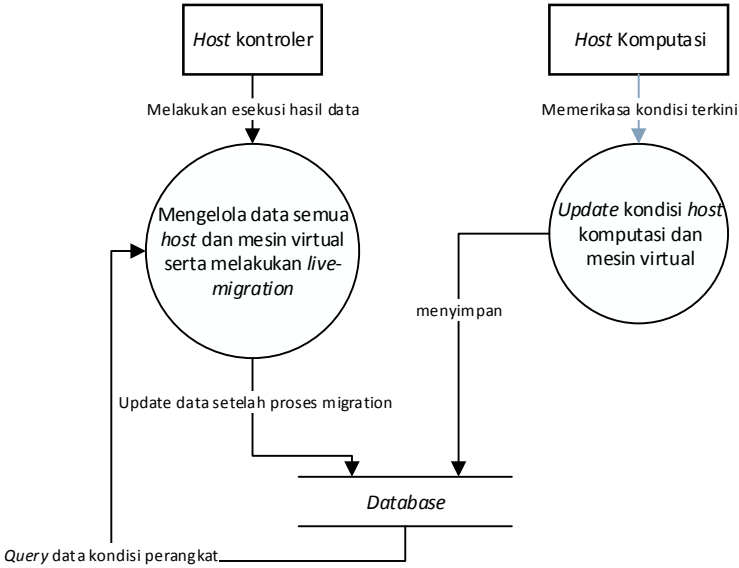
Tiga buah komputer lain sebagai perangkat komputasi (komputasi 1, komputasi 2, dan komputasi 3) yang bertugas menjadi sumber daya untuk virtualisasi. Perangkat komputasi diharuskan mendukung KVM untuk proses virtualisasi. Pada sistem ini menggunakan satu buah komputer arsitektur perangkat digambarkan pada Gambar 3.3.



Gambar 3.3 Gambaran Arsitekur Perangkat Komputasi Awan

3.4. Diagram Alir Data *Level* 0

Diagram aliran data level 0 merupakan gambaran mengenai fungsionalitas sebuah sistem. Sistem yang akan dibangun adalah sistem manajemen sumber daya komputasi awan yang dapat membagi kinerja mesin virtual yang ada pada perangkat komputasi. Sistem bertujuan untuk membagi kinerja sumber daya perangkat komputasi yang mengalami beban kerja berlebih dengan melakukan dinamik *live migration.* Berdasarkan diagram alir data pada Gambar 3.4, sistem diawali dengan mendapatkan nilai kondisi dari tiap perangkat *host* komputasi yang kemudian dimasukan ke dalam *database,* dari data tersebut nantinya akan dioleh oleh sistem yang ada pada *host* kontoler untuk mendapatkan data yang ideal pada proses *live migration*.



Gambar 3.4. Perancangan Diagram Alir Data *Level* 0

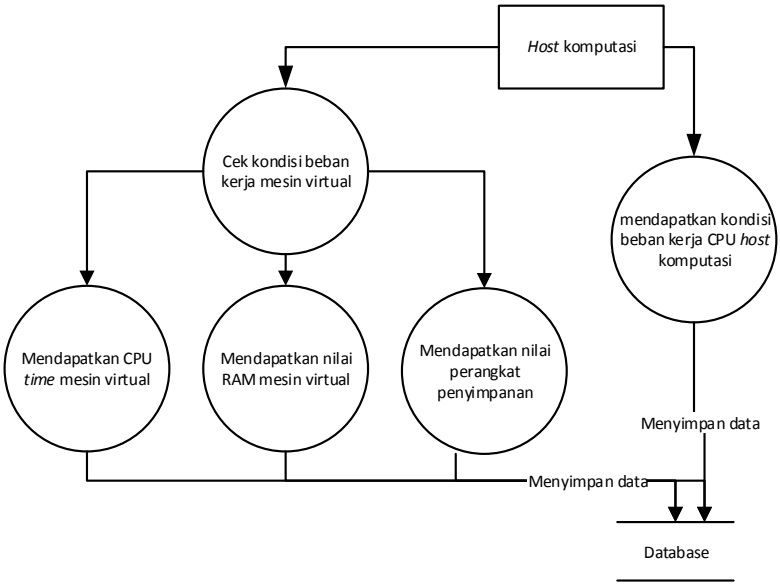
Hasil proses ini adalah melakukan *live migration* dari data yang didapat dari *query database*. Semua proses yang berjalan pada komputasi baik itu beban kerja perangkat dan mesin virtual dicatat dalam *database*. Kemudian data dari prangkat komputasi diolah oleh perangkat kontroler untuk menentukan proses *live migration* paling optimal.

3.5. Perancangan Alir Aplikasi Sistem

Alur pada setiap proses yang terdapat dalam sistem digambarkan menggunakan diagram alir data untuk memudahkan pemahaman secara garis besar proses yang terdapat dalam sistem. Diagram aliran dari sistem terdiri pengecekan kondisi *host* komputasi dan mesin virtual, mendapatkan *host* komputasi yang mengalami kelebihan beban kerja, mendapatkan mesin virtual yang memiliki *Minimum Migration Time* paling minimum, dan pengecekan tujuan *host* komputasi untuk proses *live migration*, pengalokasian mesin virtual yang tidak aktif.

3.5.1. Diagram Alir Data Pengecekan *Host* Komputasi dan Mesin Virtual

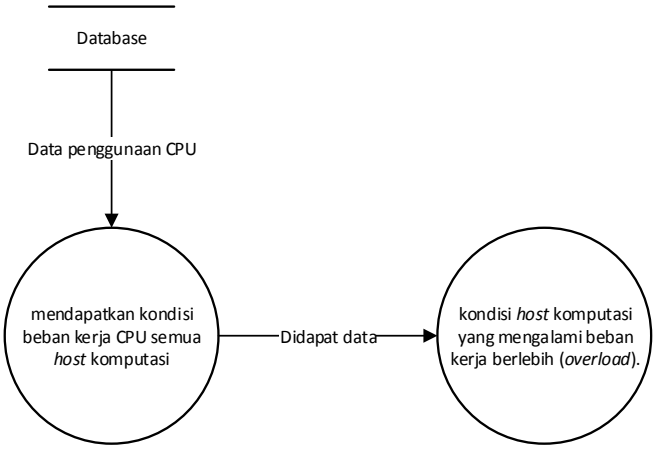
Pada *host* komputasi terdapat sistem *cloud child* yang bertugas untuk mendapatkan data dari kondisi *host* komputasi. Kondisi tersebut meliputi penggunaan beban kerja CPU dan kondisi mesin virtual. Pada pengecekan kondisi beban CPU, sistem akan memeriksa penggunaan CPU secara *real-time*, data tersebut kemudian akan diakumulasi menjadi persentase penggunaan CPU. Pada pengecekan kondisi mesin virtual, sistem akan mencatat CPU *time* mesin, uuid mesin virtual, RAM dan penyimpanan. Semua data yang didapat dari sistem perangkat kan disimpan ke dalam *database.* Diagram alir untuk pengecekan kondisi perangkat komputasi dan mesin virtual dapat dilihat pada Gambar 3.5.



Gambar 3.5 Diagram Alir Data Pengecekan Sumber Daya Perangkat Komputasi dan Mesin Virtual

3.5.2. Diagram Alir Data Mendapatkan *Host* Komputasi yang Memiliki Beban Kerja Komputasi Berlebih

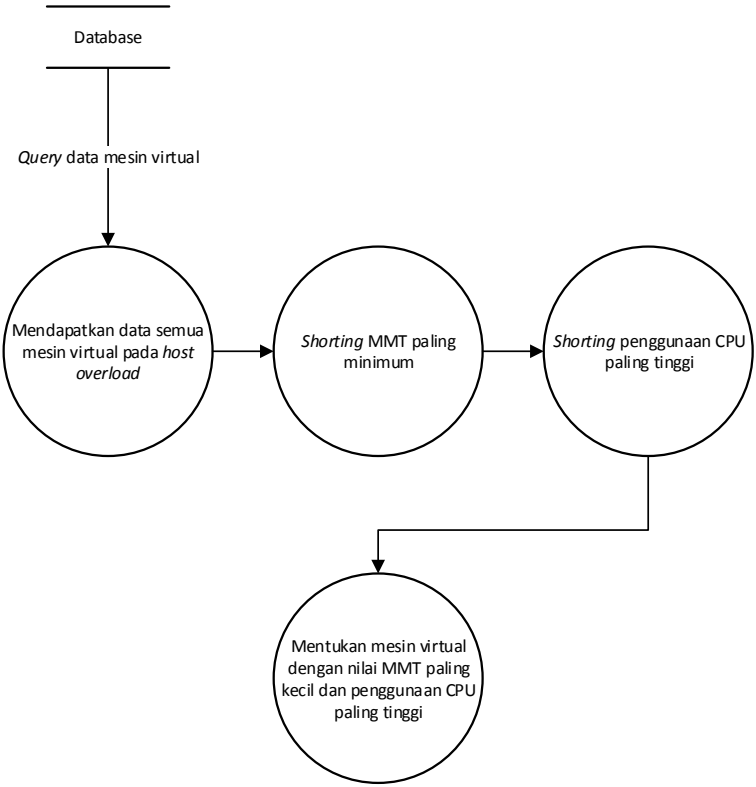
Pengecekan beban kerja komputasi berlebih (*overload*) diperlukan untuk memilih perangkat komputasi atau *host* yang memiliki beban kerja berlebih dari data penggunaan CPU. Pengecekan beban kerja komputasi berlebih dilakukan di sistem *Cloud Head* yang berjalan pada perangkat kontroler. Sistem akan memeriksa data penggunaan beban kerja dari semua *host* komputasi yang telah tercatat ke dalam *database*. Kemudian sistem akan menentukan *host* komputasi mana yang mengalami beban kerja berlebih sesuai batas nilai yang ditetapkan. Diagaram alir pengecekan *host* komputasi yang memiliki beban kerja komputasi belebih dapat dilihat pada Gambar 3.6.



Gambar 3.6 Diagram Alir Data Mendapatkan *Host* Komputasi yang Memiliki Beban Kerja Komputasi Berlebih

3.5.3. Diagram Alir Data Mendapatkan Mesin Virtual yang Akan Dipindah

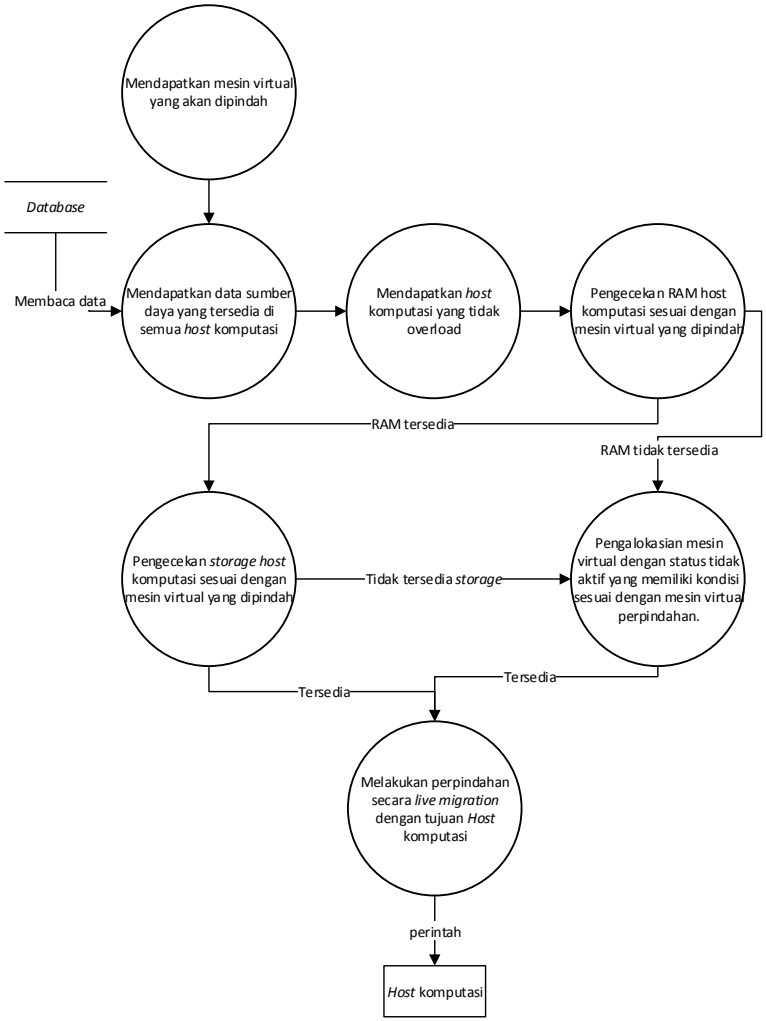
Tahap selanjutnya adalah menentukan mesin virtual yang mempunyai nilai yang ideal yang akan dipindah dengan menentukan MMT paling minimum pada mesin virtual yang terdapat pada perangkat atau *host* komputasi yang mengalami beban kerja komputasi berlebih. MMT adalah waktu yang digunakan perangkaat komputasi untuk memindahkan mesin virtual ke perangkat komputasi lain. Kemudian sistem akan mencari mesin virtual dengan penggunaan CPU paling tinggi dengan mengukur perbedaan CPU *time* sebelum pengecekan dan sesudah pengecekan pada mesin virtual. Pemilihan penggunaan CPU paling tinggi diperlukan jika terdapat MMT dengan nilaiyang sama, perpindahan mesin virtual yang memiliki beban kerja tinggi juga akan lebih mengurangi kinerja *host* yang sedang mengalami kelebihan beban kerja komputasi. Diagram alir menentukan MMT dapat dilihat di Gambar 3.7.



Gambar 3.7 Diagram Alir Data Menentukan Perangkat Komputsi yang Memiliki Beban Kerja Berlebih

3.5.4. Diagram Alir Data Menentukan Tujuan Mesin Virtual

Selah didapat mesin virtual yang akan dipindah, sistem akan menentukan tujuan perpindahan mesin virtual ke perangkat komputasi atau *host* lain. Dimulai dari pengecekan kondisi *host* komputasi yang tidak memiliki beban kerja komputasi berlebih sesuai batasan yang ditentukan.



Gambar 3.8. Diagram Alir Data Menentukan Tujuan Mesin Virtual yang Akan Dipindah

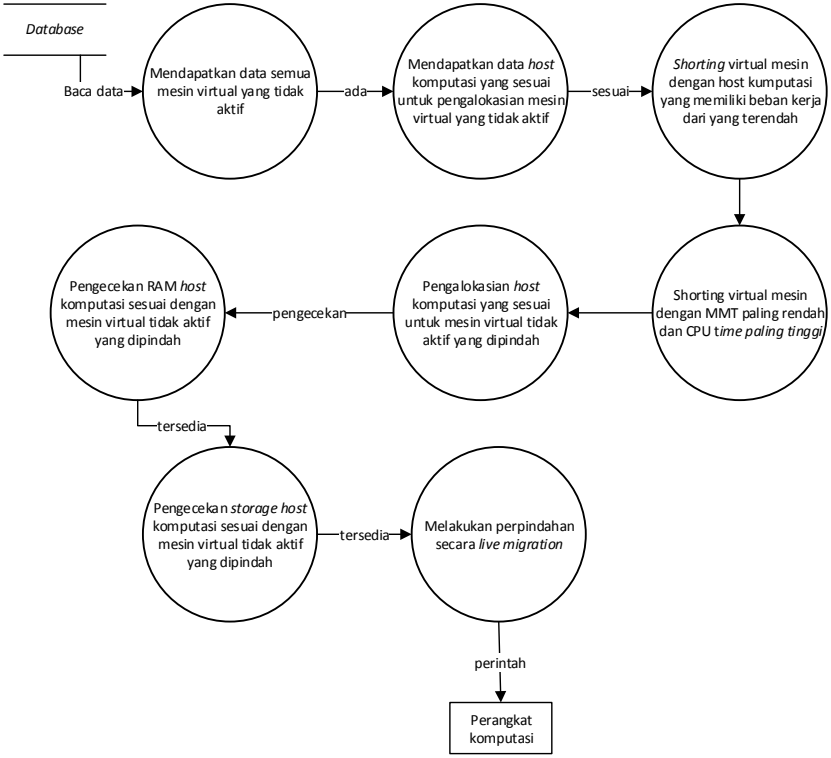
Kemudian sistem akan melakukam pemeriksaan ketersediaan RAM dan *storage* yang sesuai dengan mesin virtual yang

dipindah. Jika tersedia maka sistem akan mengalokasikan *host* komputasi lain untuk tujuan perpindahan mesin virtual dengan perangkat komputasi yang memiliki beban kerja berlebih (*overload)*. Jika tidak tersedia maka sistem akan mengalokasikan kembali tujuan perangkat komputasi dengan mengalokasikan mesin virtual yang status tidak aktif. Mesin virtual yang tidak aktif akan dipindahkan ke perangkat lain yang sesuai untuk digantikan mesin virtual yang berjalan di perangkat komputasi dengan beban kerja berlebih. Diagram alir menentukan tujuan mesin virtual yang akan dipindah dapat dilihat di Gambar 3.8.

3.5.5. Diagram Alir Data Pengalokasian Mesin Virtual yang Tidak Aktif

Pengalokasian mesin virtual yang tidak aktif dapat dijalankan jika semua perangkat komputasi tidak tersedia. Pengalokasian mesin virtual digunakan untuk mendapatkan pengalokasian baru dengan memindahkan mesin virtual yang tidak aktif untuk digantikan mesin virtual yang aktif dan memiliki MMT paling minimum yang berajalan pada perangkat komputasi yang memiliki beban kerja komputasi yang berlebih.

Dimulai dari sistem mendapatkan data mesin virtual yang tidak aktif. Kemudian sistem akan mengalokasikan tujuan perpindahan dari mesin virtual yang memiliki kondisi yang sesuai sebagai pengganti mesin virtual pada komputasi berlebih. Jika ternyata memenuhi, sistem akan mendapatkan mesin virtual pengganti MMT paling minimum. Setelah didapat maka sistem akan menentukan tujuan perpindahan ke perangkat komputasi lain dengan sumber daya yang tersedia untuk mesin virtual. Setalah didapat data perangkat komputasi yang tersedia untuk mesin virtual pengganti, sistem akan melakukan *live migration* mesin virtual pengganti ke perangkat lain sebagai pengganti alokasi mesin virtual yang mengalami kinerja komputasi berlebih. Diagram alir pengalokasian mesin virtual yang tidak aktif dapat dilihat Gambar 3.9.



Gambar 3.9. Diagram Alir Data Pengalokasian Mesin Virtual yang Tidak Aktif

3.5.6. Perancangan *Database*

Perancangan *database* diperlukan untuk menyimpan dari sistem ketika dijalankan dengan menambahkan tabel baru yang diperlukan perangkat lunak untuk berintegrasi dengan komputasi awan OpenStack. Pada Tugas Akhir ini menambahkan dua tabel yaitu tabel `host` untuk menyimpan data perangkat komputasi dan tabel `vm` untuk menyimpan data semua mesin virtual.

3.5.6.1. Tabel *Host*

Pada Gambar 3.10. adalah tabel `host` yang digunakan untuk menyimpan informasi tambahan *host* komputasi. Informasi yang disinpan adalah `id`, `host_name` untuk menyimpan nama perangkat komputasi, `host_CPU_usage` untuk menyimpan data penggunaan CPU semua *host* komputasi, `disk_gb_free` untuk menyimpan data *storage* kosong dari alokasi mesin virtual, `memory_free` digunakan untuk menyimpan data RAM kosong dari alokasi mesin virtual.

host	
host_id	**int(11) auto increment**
host_name	**varchar(100)**
disk_gb_free	**int (11)**
memory_free	**int(11)**
host_CPU_Usage	**float**

Gambar 3.10. Tabel host

3.5.6.2. Tabel VM

Pada Gambar 3.11 merupakan tabel `vm` yang dugnakan untuk menyimpan informasi mesin virtual pada perangkat komputasi. Informasi yang disinpan adalah `vm_id`, `vm_name` untuk menyimpan nama mesin virtual, `vm_uuid` digunakan untuk menyimpan uuid dari mesin virtual, `vm_RAM` digunakan untuk menyimpan data RAM darimesin virtual, `vm_storage` digunakan untuk menyimpan total alokasi penyimpanan mesin virtual, `vm_cpu_before` digunakan untuk menyimpan data CPU *time* terkahir mesin virtual, `vm_CPU_After` digunakan untuk menyimpan CPU time terbaru dari pengecekan ,

`vm_CPU_Total` digunakan untuk menyimpan data total penggunaan CPU pada satu kalipengecekan. Dari `vm_CPU_Total` dapat diperloleh data penggunaan CPU paling tinggi di mesin virtual.

VM	
VM_id	**int (11) auto increment**
VM_name	**varchar(100)**
VM_uuid	**varchar(100)**
VM_RAM	**int (11)**
VM_Storege	**int (11)**
VM_CPU_Before	**float**
VM_CPU_After	**float**
VM_CPU_Total	**float**

Gambar 3.11. Tabel VM

3.6. Rancang Antarmuka Sistem

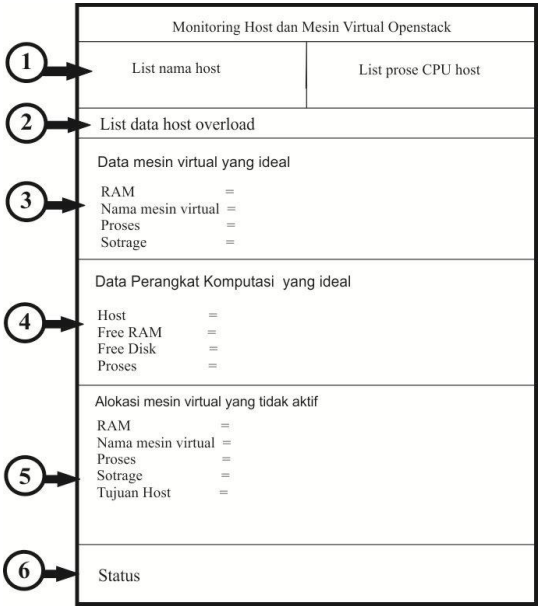
Pada Tugas Akhir ini, antarmuka sistem berupa *console*, yang menampilkan data secara langsung dari proses komputasi. Antar muka sistem dibedakan menjadi dua yaitu antarmuka sistem pusat dan sistem komputasi.

3.6.1. Rancang Antarmuka Sistem *Cloud Head*

Gambar 3.12. merupakan rancangan antarmuka untuk sistem pusat. Terdapat beberapa bagian pada tampilan pusat, adapun penjelasnya sebagai berikut:

1. Merupakan bagian menampilkan daftar dari semua *host* komputasi dan proses penggunaan CPU dalam satuan persentase.
2. Meruapakan tampilan *host* komputasi yang mengalamai beban kerja berlebih. Tampilan ini akan muncul jika didapat mesin virtual yang mengalami beban kerja berlebih.

3. Merupakan tampilan data mesin virtual yang paling ideal dari *host* komputasi yang mengalami beban kerja komputasi berlebih yang akan dipindah. Tampilan ini juga akan muncul jika didapat *host* komputasi yang mengalami beban kerja komputasi berlebih.
4. Merupakan tampilan data *host* komputasi yang mempunyai sumber daya yang sesuai untuk proses perpindahan. Tampilan ini juga akan muncul jika didapat mesin virtual yang sesuai dengan sumber daya perangkat yang tidak mengalami beban kerja berlebih.
5. Merupakan tampilan data mesin virtual yang mempunyai status tidak aktif dan tampilan tujuan alokasi perpindahan host komputasi lain yang memiliki sumber daya sesuai untuk parameter pengalokasian. Tampilan ini juga akan muncul jika tampilan bagian nomor 4 tidak didapatkan.
6. Menampilkan status perpindahan dari sistem.

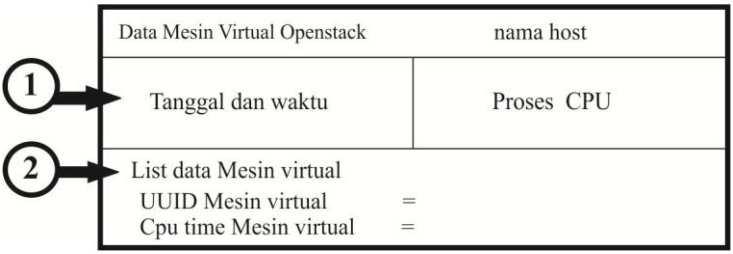


Gambar 3.12. Rancang Antarmuka Sistem *Cloud Head*

3.6.2. Rancang Antarmuka Sistem *Cloud Child*

Gambar 3.13. merupakan rancangan antarmuka untuk proses komputasi. Terdapat beberapa bagian pada tampilan pusat, adapun penjelasnya sebagai berikut:

1. Meruapakan tampilan dari penggunaan CPU beserta tanggal dan waktu alokasi penggunaan CPU.
2. Merupakan tampilan dari list mesin virtual yang ada pada Perangkat komputasi.



Gambar 3.13. Rancang Antarmuka Sistem *Cloud Child*

BAB IV
IMPLEMENTASI

Setelah melewati proses analisis dan perancangan perangkat lunak, maka dilakukan implementasi sistem. Bab ini membahas mengenai implementasi sistem yang meliputi tahap-tahap implementasi program dalam *pseudocode* dan implementasi antarmuka yang telah dibahas pada Bab III.

4.1. Lingkungan Pembangunan Sistem

Pembangunan sistem manajemen sumber daya komputasi awan pada tugas akhir ini dibangun pada lingkungan yang akan dijabarkan pada bagian selanjutnya.

4.1.1. Lingkungan Perangkat Lunak

Pembangunan aplikasi Tugas Akhir ini menggunakan bantuan perangkat lunak sebagai berikut:

- Sistem Operasi CentOS versi 6.5 64 Bit sebagai sistem operasi semua *host*.
- Glusterfs 3.5 sebagai penyedia layanan distribusi *file system* untuk membangun komputasi awan.
- QEMU sebagau dasar virtualisasi yang mendukung jalannya virtualisasi pada *hypervisor* KVM.
- KVM sebagai hypervisor yang didukung QEMU dalam menjalankan mesin virtual.
- Libvirt 2.6.20 sebagai pendukung komputasi awan untuk mengelola mesin virtual melawati pustaka API.
- Mysql sebagai RDBMS (Relational Database Manajemen System) untuk menyimpan data informasi komputasi awan
- Qpid AMQP Message Broker 0.28
- OpenStack sebagai *platform* penyedia komputasi awan.
- CirrOS 64-bit, sebagai sistem operasai mesin virtual.

4.1.2. Lingkungan Perangkat Keras

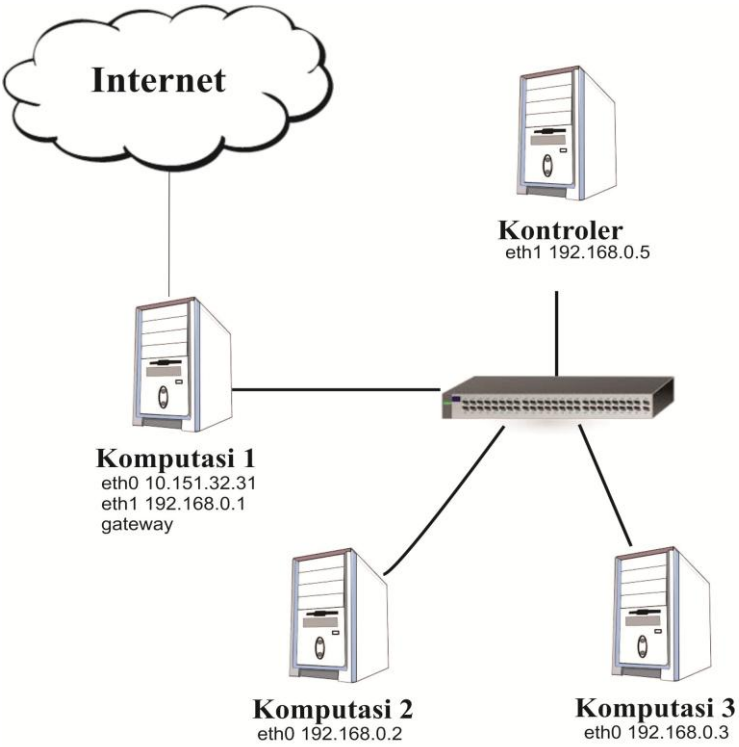
Spesifikasi perangkat keras yang digunakan untuk pembangunan sistem tugas akhir ini menggunakan empat buah komputer dan satu *swicth*. Spesifikasi dari perangkat yang digunakan adalah sebagai berikut.

- Satu komputer rakitan, dengan spesifikasi:
  - Intel (R) Core (TM) 2 Dou CPU E7400 @ 2.80GHz
  - 2 GB DDR2
  - 250 GB MAXTOR STM325031
  - RTL8111/8168/8411 PCI Express Gigabit Ethernet
- Dua komputer merk ACER
  - Intel (R) Core (TM) i3-2120 CPU @ 3.30GHz
  - 4 GB DDR3
  - 500GB WDC WD5000AAKX-2
  - Realtek RTL8111/8168B PCI Express Gigabit Ethernet
- Satu komputer merk HP
  - Intel (R) Core (TM) i3-3220 CPU @ 3.30GHz
  - 2 GB DDR3
  - 500GB WDC WD5000AAKX-6
  - Realtek RTL8111/8168B PCI Express Gigabit Ethernet
- Satu buah *switch* D-link series DES-1210-28 dengan 28 port 10/100 mbps web smart switch dengan 4 gigabit ports (2 UTP dan 2 COMBO UTP/SFP) dapat dilihat seperti Gambar 4.1

Gambar 4.1. D-link DES-1210-28

Pada tugas akhir ini menggunakan empat buah komputer. Komputer rakitan yang bertugas sebagai perangkat kontroler. Dua buah komputer Acer (komputasi 1 dan komputasi 2) dan satu komputer HP (komputasi 3) bertugas sebagai perangkat yang digunakan untuk perangkat komputasi.

Pada sistem ini menggunakan satu buah perangkat komputasi sebagai penghubung jaringan publik dan juga bertindak sebagai *gateway*. *Gateway* terhubung dengan jaringan luar lewat *network interface* eth0 dan eth1 untuk terhubung jaringan lokal.



Gambar 4.2. Arsitektur Jaringan Komputasi awan

Semua perangkat berjalan pada jaringan lokal dengan menggunakan *switch* D-link. Perangkat komputasi yang lain selain perangkat *gateway* terhubung jaringan lokal dengan eth0 sedangkan perangkat kontroler menggunakan eth1 *network interface*. Untuk mendukung akses internet dari publik, semua perangkat menggunakan Network Address Translation (NAT) dari perangkat *gateway*. Arsitektur perangkat dan jaringan digambarkan pada Gambar 4.2.

Tabel 4.1. Penamaan Host Pada Sistem

No	Jenis Komputer	Alamat IP	Fungsi Perangkat	Nama Host
1	Rakitan	192.168.0.5	Kontroler	Controller
2	Acer	10.151.32.31 192.168.0.1	Komputasi	Compute1
3	Acer	192.168.0.2	Komputasi	Compute2
4	HP	192.168.0.3	Komputasi	Compute3

Pada Tabel 4.1 merupakan tabel penamaan semua *host* yang terdapat didalam sistem komputasi awan. Penamaan perangkat bertujuan untuk memudahkan memeriksa data dan melakukan operasi ke perangkat.

4.2. Implementasi Data *Training*

Implementasi data *training* adalah implementasi mendapatkan data yang digunakan sebelum implementasi perangkat lunak. Implementasi data *training* diperlukan untuk mendapatkan data faktor-faktor yang mempengaruhi *Minimum Migration Time* (MMT) dan mengukur berapa lama jeda *host* komputasi mendapatkan nilai beban kerja komputasi yang stabil setelah proses perpindahan atau *live-migration*.

4.2.1. Menghitung Nilai *Minimum Migration Time*

Implementasi data *training* untuk menentukan faktor-faktor yang mempengaruhi MMT dilakukan dengan membuat suatu program sederhana untuk mengukur berapa lama waktu untuk sekali perpindahan ke *host* lain. Percobaan dilakukan dengan memberikan nilai RAM, *storage*, beban kerja CPU, VCPU, dan menambahkan *additional storage* dari OpenStack-Nova-Volume yang berbeda pada mesin virtual. Setelah itu melakukan perpindahan dari mesin virtual yang telah ditentukan.

Semua aktifitas proses komputasi awan dan mesin virtual dicatat dalam *data base* OpenStack. Salah satunya adalah ketika *live-migration*. Sehingga program dibuat untuk melakukan pengecekan mesin virtual dari status aktif berubah menjadi *migrating* dan kembali lagi menjadi aktif. Status *migrating* pada mesin virtual ditampilkan ketika mesin virtual sedang dalam proses perpindahan dan akan berubah menjadi aktif jika proses perpindahan selsesai. Proses terebut membutuhkan waktu, sehingga dalam pengambilan data dibuat program untuk menghitung berapa lama mesin virtual melakukan perpindahan, adapun *pseudocode* ditunjukan pada Gambar 4.3. Program dimulai dengan melakukan pengecekan mesin virtual dengan status *migrating* pada *data base* OpenStack. Pengujian faktor-faktor yang mempengaruhi perpindahan dimualai dari RAM, VCPU, *storage*, *additioanal storage* yaitu Nova-volume, dan beban kerja pada mesin virtual.

Mengukur Waktu Perpindahan Mesin Virtual

Input: -

```
1  Inisialisasi Database OpenStaack
2  while true:
3   data ⇦ VM dengan state “migrating”
4   if(data!= null):
5      waktu=waktu+1
6   else:
7       Print Waktu yang dibutuhkan + waktu
8       Break
```

```
9    Time.sleep(1)
Output: Waktu perpindahan
```

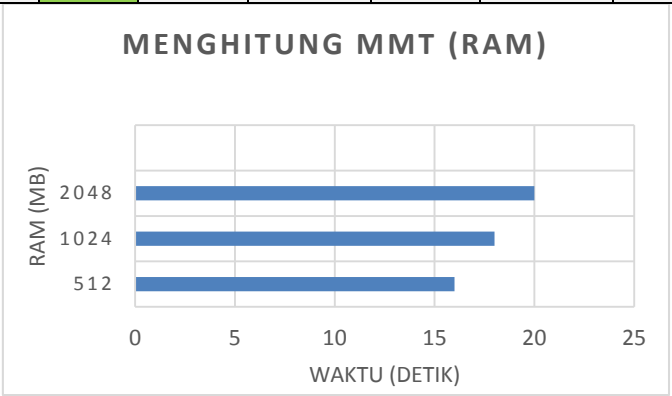
Gambar 4.3. *Pseudocode* Mengukur Waktu Perpindahan

4.2.1.1. Menghitung MMT Dengan Faktor RAM

Pengambilan data dimulai dengan memberikan nilai RAM yang berbeda pada mesin virtual VM1 512MB, VM2 1024, dan VM3 2048 dengan nilai fakor yang lain diatur sama seperti pada Tabel **4.2** . Dari data tersebut dapat disimpulkan bahwa RAM mempengaruhi lama tidaknya prosesperpindahan sesperti pada Gambar 4.4.

Tabel 4.2. Menghitung MMT Dengan Faktor RAM

VM	RAM (MB)	VCPU	Storage (GB)	Nova Volume	Beban Kerja (%)	Waktu (detik)
VM1	512	1	0	0	0	16
VM2	1024	1	0	0	0	18
VM3	2048	1	10	0	0	20



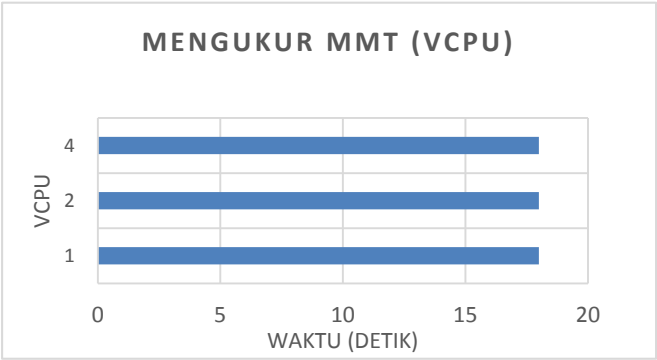
Gambar 4.4. Grafik MMT Dengan Faktor RAM

4.2.1.2. Menghitung MMT Dengan Faktor VCPU

Pengambilan data dilanjutkan dengan memberikan nilai VCPU yang berbeda pada mesin virtual VM4 1 CPU, VM5 2 VCPU, dan VM6 4 VCPU dengan nilai fakor yang lain diatur sama seperti pada Tabel 4.3. Dari data tersebut dapat disimpulkan bahwa VCPU tidak mempengaruhi lama tidaknya proses perpindahan seperti digambarkan pada grafik Gambar 4.5.

Tabel 4.3. Mengukur MMT dengan faktor VCPU

VM	RAM (MB)	VCPU	Storage (GB)	Nova Volume	Beban Kerja (%)	Waktu (detik)
VM4	1024	1	0	0	0	18
VM5	1024	2	0	0	0	18
VM6	1024	4	0	0	0	18



Gambar 4.5. Grafik MMT Dengan Faktor VCPU

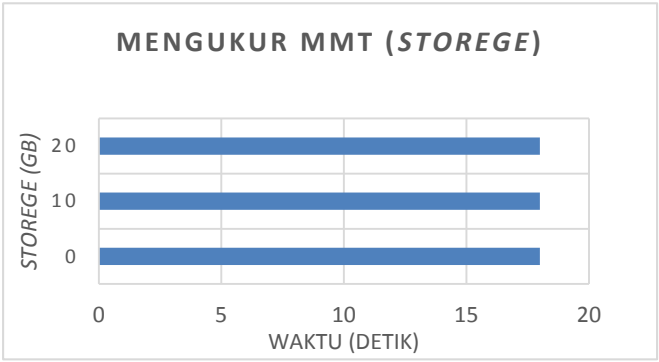
4.2.1.3. Menghitung MMT Dengan Faktor *Storage*

Pengambilan data dilanjutkan dengan memberikan nilai *storage* yang berbeda pada mesin virtual VM7 0 GB, VM5 10 GB,

dan VM6 20GB dengan nilai fakor yang lain diatur sama seperti pada Tabel 4.4. Dari data tersebut dapat disimpulkan bahwa *storage* tidak mempengaruhi lama tidaknya proses perpindahan seperti digambarkan pada grafik Gambar 4.6.

Tabel 4.4. Menghitung MMT Dengan Faktor *Storage*

VM	RAM (MB)	VCPU	Storage (GB)	Nova Volume	Beban Kerja (%)	Waktu (detik)
VM7	1024	1	0	0	0	18
VM8	1024	1	10	0	0	18
VM9	1024	1	20	0	0	18



Gambar 4.6. Grafik MMT Dengan Faktor *Storage*

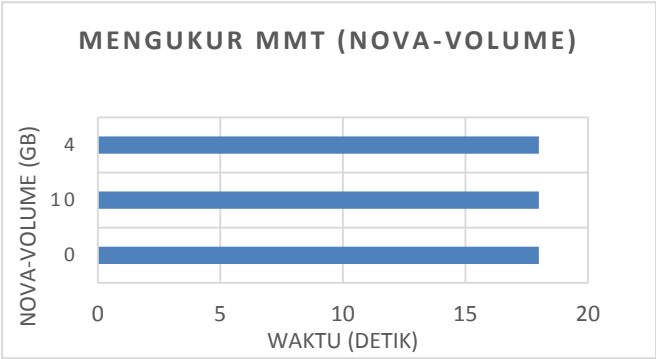
4.2.1.4. Mengukur MMT Dengan Faktor Nova-Volume

Selanjutnya pengambilan data dilanutkan dengan memberikan nilai *storage* Nova-Volume yang berbeda pada mesin virtual VM10 0 GB, VM11 10 GB, dan VM12 20GB dengan nilai fakor yang lain diatur sama seperti pada Tabel 4.5. Dari data tersebut dapat disimpulkan bahwa *storage* tidak

mempengaruhi lama tidaknya proses perpindahan seperti digambarkan pada grafik Gambar 4.7.

Tabel 4.5. Pengujian Dengan Faktor OpenStack-Nova-Volume

VM	RAM (MB)	VCPU	Storage (GB)	Nova Volume	Beban Kerja (%)	Waktu (detik)
VM10	1024	1	0	0	0	16
VM11	1024	2	0	10	0	18
VM12	1024	4	0	20	0	20



Gambar 4.7. Grafik MMT Dengan Faktor Nova-Volume

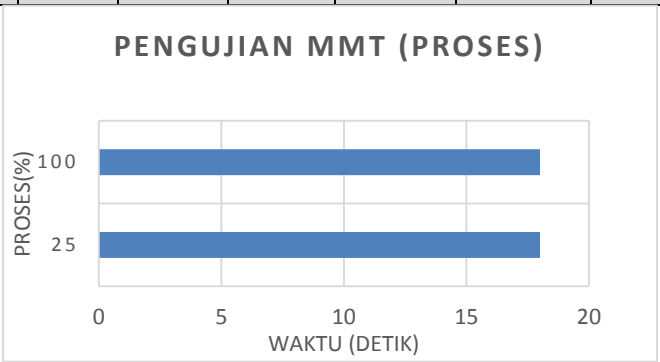
4.2.1.5. Pengujian MMT Dengan Faktor Beban Kerja

Selanjutnya pengambilan data dilanjutkan dengan memberikan beban kerja pada mesin virtual dengan nilai beban kerja yang berbeda. Beban kerja pada mesin virtual VM13 0%, dan VM14 100% dengan nilai fakor yang lain diatur sama seperti

pada Tabel 4.6. Dari data tersebut dapat disimpulkan bahwa *storage* tidak mempengaruhi lama tidaknya proses perpindahan seperti digambarkan pada grafik Gambar 4.8.

Tabel 4.6. Data Pengujian Dengan Faktor Beban Kerja

VM	RAM (MB)	VCPU	Storage (GB)	Nova Volume	Beban Kerja (%)	Waktu (detik)
VM13	1024	1	0	0	0	18
VM14	1024	1	0	0	100	18



Gambar 4.8. Pengujian MMT Dengan Faktor Proses

Dari lima percobaan untuk mencari faktor MMT didapatkan besar kecil nya nilai RAM adalah faktor yang paling mempengaruhi waktu pada proses perpindahan, adapun kesimpulanya dapat dilihat pada Tabel 4.7

Tabel 4.7. Kesimpulan Faktor yang Mempengaruhi Perpindahan

Faktor	Mempengaruhi
RAM	Ya
VCPU	Tidak
Storage	Tidak
Strorage Nova-Volume	Tidak

Faktor	Mempengaruhi
Proses	Tidak

4.2.2. Penentuan Jeda Waktu yang Ideal Untuk Menstabilkan Beban CPU Setelah Proses Perpindahan

Perpindahan mesin virtual untuk mengurangi beban kerja *host* yang overload ternyata membutuhkan waktu bagi *host* untuk menstabilkan beban kerja CPU dari kelebihan beban kerja komputasi menjadi normal. Lama proses perubahan beban kerja *host* setelah proses perpindahan dapat mempengaruhi sistem untuk melakukan pengecekan dan perpindahan mesin virtual selanjutnya. Belum stabilnya atau belum berubahnya data beban CPU secara langsung dapat membuat sistem melakukan pengulangan perpindahan. Misalnya *host* komputasi1 memiliki beban kerja komputasi berlebih, dalam kurun waktu tertentu *host* komputasi1 ternyata masih memiliki beban kerja berlebih padahal proses mesin virtual telah dipindahkan. Hal ini menjadikan penambahan jeda pada setiap pengecekan beban kerja komputasi diperlukan. Maka dari itu implementasi data training untuk mendapat nilai lama waktu bagi *host* setelah proses perpindahan diperlukan. Pengambilan data dimulai dengan melakukan perpindahan VM dari *host* yang memiliki beban kerja berlebih ke *host* lain secara manual. Proses penentuan jeda dimulai ketika mesin virtual yang telah dipindahkan, kemudian dilakukan pengecekan beban kerja CPU dari waktu ke waktu setelah proses perpindahan. Data dari penentuan stabilitas CPU dapat dilihat pada Tabel 4.8.

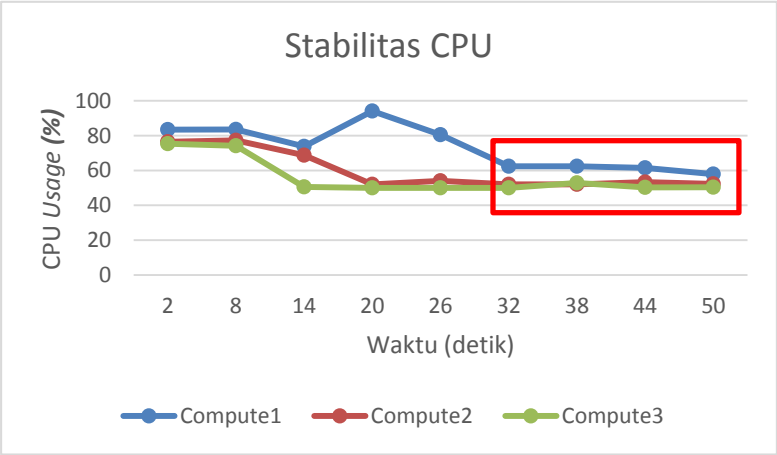
Tabel 4.8. Penentuan Waktu Jeda Untuk Mengukur Stabilitas CPU Setelah Perpindahan

No	Asal *Host*	Kondisi Awal (%)	Waktu (detik)	Kondisi Akhir (%)
1	Compute1	83.31	2	83.43
			4	93.98
			6	93.44
			8	83.56
			10	83.32
			12	92.84
			14	73.75
			16	81.67
			18	86.41
			20	94.11
			22	92.41
			24	91.53
			26	80.46
			28	76.38
			30	67.83
			32	62.38
			34	83.68
			36	62.92
			38	62.37
			40	61.07
			42	61.03
			44	61.5
			46	64.92
			48	62.81
			50	57.92
2	Compute2	76.31	2	76.43
			4	76.23
			6	76.94
			8	77.32
			10	76.03
			12	79.84
			14	68.75

No	Asal *Host*	Kondisi Awal (%)	Waktu (detik)	Kondisi Akhir (%)
			16	55.67
			18	51.41
			20	52.00
			22	51.31
			24	51.03
			26	54.03
			28	51.78
			30	51.48
			32	52.01
			34	51.31
			36	51.09
			38	52.02
			40	51.03
			42	51.01
			44	53.31
			46	51.29
			48	50.05
			50	52.33
3	Compute3	77.59	2	75.35
			4	79.23
			6	85.12
			8	74.12
			10	83.03
			12	50.89
			14	50.54
			16	51.51
			18	50.89
			20	50.00
			22	50.31
			24	50.03
			26	50.03
			28	50.78
			30	50.48
			32	50.01
			34	50.31

No	Asal *Host*	Kondisi Awal (%)	Waktu (detik)	Kondisi Akhir (%)
			36	50.79
			38	52.82
			40	50.03
			42	50.01
			44	50.31
			46	50.29
			48	50.15
			50	50.33

Dari pengumpulan data pada Tabel 4.8. diperoleh kesimpulan bahwa sistem memerlukan jeda setiap kali pengecekan kondisi beban kerja CPU setelah detik ke 32. Data jeda waktu yang ideal dapat disimpulkan pada grafik di Gambar 4.9. Dari grafik tersebut didapatkan *host* compute1 yang bertugas sebagai perangkat komputasi dan *gateway* membutuhkan waktu lebih lama daripada *host* lain untuk menstabilkan beban kerja CPU dari kondisi kelebihan beban kerja ke kondisi noramal. Nilai yang ideal ditunjukan kotak merah pada Gambar 4.9 yaiutu nilai diatas 32 detik.



Gambar 4.9. Grafik Stabilitas CPU Setelah Proses Perpindahan.

4.3. Implementasi Perangkat Lunak

Implementasi sistem manajemen sumber daya terbagi menjadi dua yaitu *Cloud Child*, dan *Cloud Head*. Setiap bagian dari implementasi akan dijelaskan lebih lanjut pada setiap subbab.

4.3.1. Implementasi *Cloud Child*

Gambar 4.10 menunjukkan *pseudocode* implementasi *Cloud Child* berdasarkan diagram alir dari bab sebelumnya. Proses ini akan mengeluarkan keluaran informasi kondisi perangkat komputasi dan kondisi mesin virtual pada perangkat komputasi.

Prosedur 1. Pengecekan Kondisi *Host* dan mesin virtual

```
Input: -
1   Inisialisasi Database
2   Inisialisasi libvirt
3   Interval time = 2 detik
4   while true
5     FOR sebanyak jumlah mesin virtual
6       CpuTimeVm ⇦ data cpu time vm dari libvirt.info
7       Query ⇦ cek data ketersediaan vm database
8       IF Query = true
9          Update data kondisi vm pada database
10      ELSE
11         Insert data vm baru dan kondisi vm
12         pada database
13    CPUUsage ⇦ getCPU_usage selama (interval)
14    Update data CPU perangkat pada database
15
Output: Informasi kondisi perangkat komputasi dan vm
```

Gambar 4.10 *Psedocoude Cloud Child*

Penjelasan dari *pseudocode Cloud Child* adalah sebagai berikut.

1. Melakukan inisialisasi *database*, yang digunakan untuk membaca dan menulis data pada *database*.

2. Melakukan inisialisasi pada fungsi libvirt yang digunakan untuk mengakses API dari libvirt.
3. Melakukan perulangan dan pengecekan sesuai jumlah dari mesin virtual yang ada pada perangkat komputasi.
4. Mengakses API dari libvirt untuk mendapatkan *cpu time* yang digunakan untuk parameter penggunaan cpu pada mesin virtual.
5. Melakukan *query* data untuk pengecekan mesin virtual yang tersedia, jika data tidak tersedia maka akan ditambahkan ke *database*.
6. Pengecekan kondisi beban kerja *host* dengan interval waktu per dua detik sekali.
7. Pembaruan kondisi data mesin virtual terbaru.

4.3.2. Implementasi *Cloud Head*

Implementasi C*loud Head* terabagi menjadi empat proses bagian yaitu mendapatkan nilai perangkat yang mengalami beban kerja komputasi berlebih seleksi mesin virtual yang akan dipindah, seleksi perangkat komputasi untuk perpindahan, pengalokasian mesin virtual yang tidak aktif. Setiap bagian akan dijelaskan lebih lanjut pada subbab berikut.

a. Seleksi Untuk Menentuan Host Komputasi yang Mengalami Beban Kerja Komputasi Berlebih

Prosedur 2. Penentuan Host Komputasi yang mengalamai beban kerja komputasi berlebih.	
Input: Data perangkat komputasi	
1	Inisialisasi Database
2	Threshold = 100 – (100/n VCPU)
3	Perangkat ⇦ data semua perangkat komputasi
4	Overload ⇦ perangkat yang memiliki beban kerja > Threshold
5	Sorting berdasarkan CPU *Usage pada* Overoad yang dari yang paling besar
6	Time.sleep(32)

7	`return overload`
`Output: Informasi perangkat komputasi yang overload.`	

Gambar 4.11. *Pseudocode* Dari Host Dengan Beban Kerja Komputasi Overload

Pada Gambar 4.11. merupakan *psedocoude* untuk menjelaskan bagaimana mendapatkan data perangkat komputasi yang berlebih dengan mengambil data dari *database*. Penjelasan dari *pseudocode* adalah sebagai berikut.

1. Mendapatkan data perangkat komputasi dari *database*
2. Membandingkan apakah komputasi mengalami beban kerja melebihi *threshold* atau batasan. *Threshold* diperolah dari nilai maksimal dikurangi nilai maksimal dibagi banyaknya VCPU pada *host.* Untuk lebih jelasnya menentukan *threshold* dapat dilihat pada Persamaan (1). Pemberian batasan kurang dari 100% dimaksudkan untuk membatasi beban kerja agar *host* tidak bekerja secara penuh. Dikarenakan *host* yang bekerja pada nilai 100% akan membuat kinerja *host* menjadi lambat.

   Threshold = 100% - (100% / banyaknya_VCPU) ***(1)***

3. Jika ada perangkat komputasi yang memiliki beban kerja berlebih maka data akan disimpan dalam *database*.
4. *Shorting* berdasarkan penggunaan CPU *host* paling besar yang digunakan untuk prioritas perpindahan.
5. Sistem menentukan jeda waktu seperti pada time.sleep(32) yang artinya sistem dapat melakukan pengecekan berulang setelah menit ke 32.
6. Mendapatkan data perangkat komputasi yang mengalamai beban kerja komputasi berlebih.

b. Menentukan Mesin Virtual yang Akan Dipindah

Prosedur 3. Menentukan mesin virtual yang meiliki nilai ideal untuk dipindah.

```
Input: Data perangkat komputasi yang overload
1  MesinVirtual ⇦ data semua mesin virtual yang ada
                  pada perangkat komputasi yang
                  overload
2  MMMT ⇦ mendapatkan data dari MesinVIrtual yang
          memiliki RAM paling rendah.
3  VMPindah ⇦ mendapatkan mesin virtual dengan CPU time
          paling randah data dari data MMT pada proses
          sebelumnya.
4  return VMPindah
Output: VMPindah
```

Gambar 4.12. *Pseudocode* mesin virtual yang akan dipindah

Pada Gambar 4.12 merupakan psedocoude untuk menjelaskan bagaimana mendapatkan data mesin virtual yang memiliki nilai paling ideal untuk dipindah dari perangkat yang memiliki beban kerja berlebih. Penjelasan *pseudocode* adalah sebagai berikut.

1. Membaca semua data mesin virtual yang terdapat pada *host* yang mengalamai beban kerja komputasi berlebih.
2. Memilih mesin virtual yang mempunyai *Minimum Migration Time* terkecil. Berdasarkan data percobaan yang ditunjukan pada subbab 4.2 tentang data *training* penentuan MMT, faktor utama yang mempengaruhi waktu migrasi adalah RAM. Untuk itu pada tahap ini menentukan nilai RAM paling kecil.
3. Setelah didapat mesin virtual yang dengan RAM paling kecil, maka kemudian pemilihan lanjut dengan parameter lain yaitu mesin virtual yang memiliki beban CPU paling tinggi yang dihitung dari CPU *time*.
4. Didapat data mesin virtual yang akan dipindah.

c. Menentukan tujuan perangkat komputasi untuk perpindahan mesin virtual

Pada Gambar 4.13 merupakan *pseudocode* untuk menjelaskan bagaimana mendapatkan data tujuan perangkat

komputasi yang ideal untuk perpindahan mesin virtual yang akan dipindah. Penjelasan *pseudocode* sebagai berikut.

1. Mendapatkan data RAM, UUID, dan penyimpanan dari mesin virtual yang dipindah.
2. Mendapatkan data dari semua perangkat komputasi yang tidak memiliki beban kerja melebihi batas `threshold_tujuan`. `threshold_tujuan` adalah jarak batas aman dari host tujuan dengan *host* yang overload. Diberikannya threshold_tujuan bertujuan untuk mencegah *host* tujuan tetap tidak *overload* ketika mendapat pembagian kinerja dari host yang *overload*. Sehingga mencegah sistem melakukan perpindahan berulang kali tanpa berhenti. Nilai batas threshold_tujuan didapat dari perhitungan nilai batas *threshold* untuk *host* yang mengalami beban kerja berlebih dikurangi jumlah total presentase dibagi jumlah core yang tersedia pada perangkat, seperti pada Tugas Akhir ini menggunakan tiga perangkat atau host yang digunakan untuk komputasi menggunakan CPU sebanyak 4. Sehingga nilai threshold_tujuan diperoeh seperti pada persamaan.

$$Threshold\_tujuan = Threshold - (100\% / 4) \quad \textbf{(2)}$$

3. Mengurutkan Data perangkat berdasarkan penggunaan CPU terkecil.
4. Membaca semua perangkat yang telah di urutkan.
5. Memeriksa apakah RAM dan sisa penyimpanan dari perangkat sesuai dengan perangkat. Jika tersedia, maka proses perpindahan dengan perangkat tujuan dapat dilaksanakan. Jika tidak sistem akan mengecek ke tahap selanjutnya memeriksa mesin virtual yang tidak aktif.

Prosedur 4. Menentukan tujuan perangkat komputasi untuk perpindahan mesin virtual.

```
Input: VMPindah
1  core = jumlah_total CPU
2  threshold_tujuan = threshold – (100/n CPU)
```

```
3   RAM = RAM.VMPindah
4   Flag=false
5   UUID=UUID.VMPindah
6   Penyimpanan = storage.VMPindah
7   DataPerangkat ⇦ data semua perangkat komputasi yang
8                   mempunyai     CPU     Usage     <=
                    Threshold_tujuan
9   Sorting DataPerangkat berdasarkan CPU terkecil
10   FOR x banyaknya DataPerangkat
11     IF RAM.DataPerangkat > RAM
12       IF Penyimpanan.DataPerangkat > penyimpanan
13          Perangkat_tujuan=DataPerangkat
14          Flag = true
15          Break;
16  IF Flag=true
17   Live-migration + UUID + Perangkat tujuan
18  ELSE
     Cek mesin virtual yang tidak aktif
Output: live-migration VMPindah ke perangkat komputasi
tujuan
```

Gambar 4.13. *Pseudocode* Menentukan Host Komputasi

d. Mengalokasikan Mesin Virtual dengan Status Tidak Aktif

```
Prosedur 5. Mengalokasikan mesin virtual dengan status
tidak aktif
Input: VMPindah
1   Inisialisasi database
2   DataSuspend ⇦ data semua mesin virtual dengan status
                  tidak aktif
3   RAM=RAM.VMPindah
4   Storage=Storage.VMPindah
5   Flag=false
6   Sorting DataSuspend berdasarkan perangkat komputasi
    dengan penggunaan CPU terkecil dan tidak overload

    FOR x,banyaknya DataSuspend,x++{
7    Host  ⇦  data  semua  perangkat  komputasi  kecuali
8          perangkat komputasi yang ada DataSuspend[x].
9      FOR y,banyaknya host, y++{
10      IF FreeRAM.Host[y] + RAM.DataSuspend[x] > RAM{
```

```
11          IF FreeStorage.Host[y] +
                Storage.DataSuspend[x] > Storage{
12               Perangkat_tujuan=DataPerangkat
13               Flag = true
14               Suspend=UUID.DataSuspend[x]
15               HostPindah=host[y]
                 Break;}}}
16  }
17  IF Flag=true
18   Nova resume UUID
     Live-migration + UUID + Perangkat tujuan
19   Nova suspend UUID
20  ELSE
21   Tidak ada perpindahan vesin virtual
22
Output: live-migration VMPindah ke perangkat komputasi
tujuan
```

Gambar 4.14 Alokasi mesin virtual dengan status tidak aktif

Pada Gambar 4.14 merupakan *pseudocode* untuk menjelaskan bagaimana mengalokasikan mesin virtual yang tidak aktif menjadi alokasi untuk mesin virtual yang dipindah. Penjelasan *pseudocode* dari Gambar 4.14 adalah sebagai berikut.

1. Mendapatkan data semua mesin virtual yang tidak aktif.
2. Mendapatkan data RAM dan *storage* dari mesin virtual yang akan dipindah pada perangkat komputasi dengan beban kerja berlebih.
3. Mendapatkan data mesin virtual yang tidak aktif dengan perangkat komputasi yang tidak mengalami beban kerja berlebih.
4. Mengurutkan data mesin virtual yang tidak aktif berdasarkan penggunaan CPU
5. Membaca banyaknya data dari mesin virtual dengan status tidak aktif.
6. Membaca data semua perangkat komputasi dan kemudian membandingkan apakah alokasi mesin virtual yang tidak aktif memilki RAM yang sesuai. Untuk lebih jelasnya dapat dilihat pada persamaan (3).

$$FreeRAM.Perangkat + RAM.TidakAktif > RAM.Pindah \quad \textbf{(3)}$$

7. Membaca data perangkat komputasi dan kemudian membandingkan apakah alokasi mesin virtual yang tidak aktif memilki penyimpanan yang sesuai. Untuk lebih jelasnya dapat dilihat pada persamaan (4).

$$FreeStorage.Perangkat + Storage.TidakAktif > Storage.Pindah \quad \textbf{(4)}$$

8. Jika tahap 5 dan 6 terpenuhi maka proses *live-migration* dapat dijalankan untuk mengalokasikan perangkat yang tidak aktif.
9. Setelah tahap ini selsai maka sistem akan kembali ke prosedur empat.

4.4. Implementasi Komputasi Awan

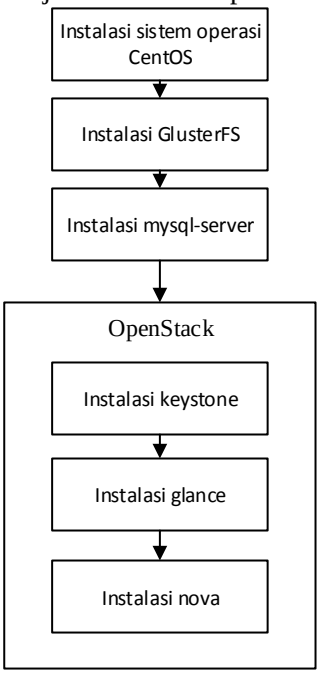
Pada subbab ini akan dibahas mengenai implementasi komputasi awan menggunakan OpenStack. Implementasi dilakukan di semua perangkat komputasi dan perangkat kontroler. Implementasi untuk perangkat kontroler berbeda dengan perangkat komputasi, perangangkat kontroler digunakan untuk mengatur semua layanan komputasi awan, sedangkan perangkat perangkat komputasi digunakan untuk penyedia sumber daya untuk virtualisasi. Berikut langkah-langkah yang digunakan untuk membangun komputasi awan menggunakan Openstack.

4.4.1. Implementasi Perangkat Kontroler

Implementasi kontroler dilakukan di perangkat kontroler. Dengan beberapa pemasangan perangkat lunak untuk mendukung proses kontroler komputasi awan.

Pada Gambar 4.15, digambarkan tahap-tahap utama untuk implementasi perangkat kontroler. Pada tiap tahap terdpat konfigurasi agar perangkat lunak atau perangkat pendukung bisa

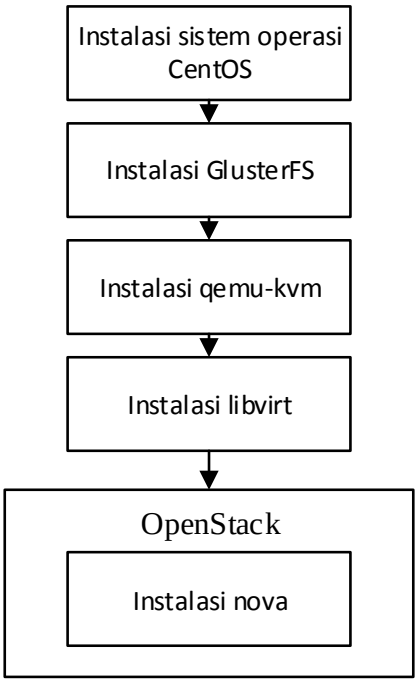
menjadi pendukung jalannya komputasi awan. Konfigurasi pada tiap tahap dijelaskan dijelaskan di Lampiran bagian A.



Gambar 4.15. Tahap Instalasi Kontroler

4.3.2. Implementasi Perangkat Komputasi

Implementasi kontroler dilakukan di tiga perangkat komputasi. Dengan beberapa pemasangan perangkat lunak untuk mendukung proses komputasi komputasi awan. Gambaran secara umum implementasi dapat dilihat pada pada Gambar 4.16. Pada tiap tahap terdapat konfigurasi agar perangkat lunak atau perangkat pendukung bisa menjadi pendukung jalannya komputasi awan. Konfigurasi pada tiap tahap dijelaskan pada Lampiran bagian B.



Gambar 4.16. Tahap Instalasi Perangkat Komputasi

4.3.2. Implementasi Perangkat *Gateway*

Implementasi perangakt g*ateway* dilakukan di perangkat komputasi1. Dengan beberapa pemasangan perangkat pendukung sama dengan perangkat komputasi lain, tetapi diberlakukan penambahan pemasangan perangkat lain dan pengaturan yang lain sebagai *gateway*. Gambaran umum pemasangan perangkat *gateway* dapat dilihat pada Gambar 4.17. Pemasangan perangkat dan pengaturan untuk perangkat *gateway* dijelaskan pada Lampiran bagian C.

Gambar 4.17. Tahap Installasi Pada Perangkat *Gateway*

4.5. Implementasi Antarmuka Pengguna

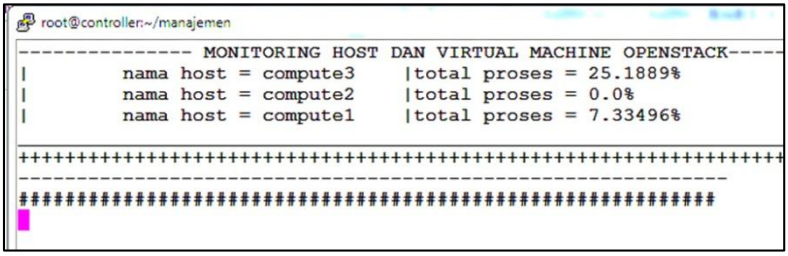
Pada subbab ini menampilkan secara umum antarmuka pengguna sistem dan komputasi awan. Pada komputasi awan menggunakan OpenStack-dashboard sedangkan pada sistem menggunakan tampilan *console*.

4.5.1. Antarmuka Sistem *Cloud Head*

Tampilan utama dari sistem *Cloud Head* terdiri dari empat bagian yaitu menampilkan data penggunaan CPU pada perangkat komputasi, menampilkan data *host* yang mempunyai beban kerja komputasi berlebih, menampilkan data mesin virtual yang akan dipindah, menampilkan perangkat komputasi sebagai tujuan perpindahan mesin virtual, menampilkan proses *live migration* pada komputasi, menampilkan data alokasi mesin virtual yang tidak aktif.

4.5.1.1. Antarmuka Monitoring Penggunaan CPU Perangkat Komputasi

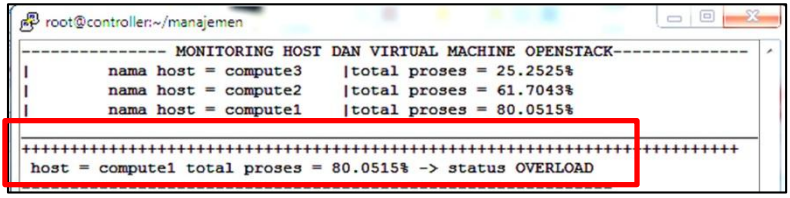
Pada Gambar 4.18 adalah tampilan data penggunaan CPU semua perangkat dalam presentase, pada tampilam tersebut menampilkan nama perangkat komputasi yang tersedia pada sistem komputasi awan.



Gambar 4.18. Antarmuka *Cloud Head*

4.5.1.2. Antarmuka Perangkat Komputasi yang memiliki Beban Kerja Berlebih

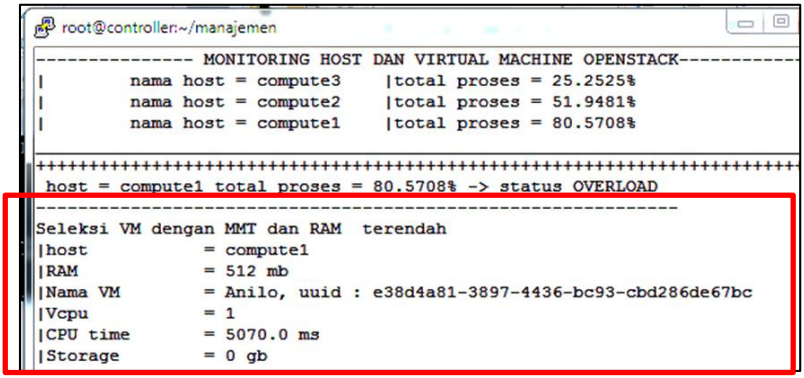
Kotak merah pada Gambar 4.19. adalah tampilan data perangkat yang mengalami beban kerja berlebih. Tampilan ini dapat ditampilkan ketika terdapat perangkat yang memiliki beban kerja berlebih.



Gambar 4.19. Antarmuka Sistem Manajemen Sumber Daya Ketika Terdapat Beban Kerja berlebih

4.5.1.3. Antarmuka Mesin Virtual yang Mempunyai Nilai Ideal Untuk Perpindahan

Kotak merah pada Gambar 4.20. adalah tampilan mesin vitual yang memiliki nilai yang ideal untuk dipindah. Tampilan ini dapat ditampilkan ketika terdapat perangkat yang memiliki beban kerja berlebih. Pada tampilan ini menampilkan data nama perangkat komputasi, alokasi RAM pada mesin virtual, nama mesin virtual, penggunaan virtual CPU, CPU time dan *storage.*

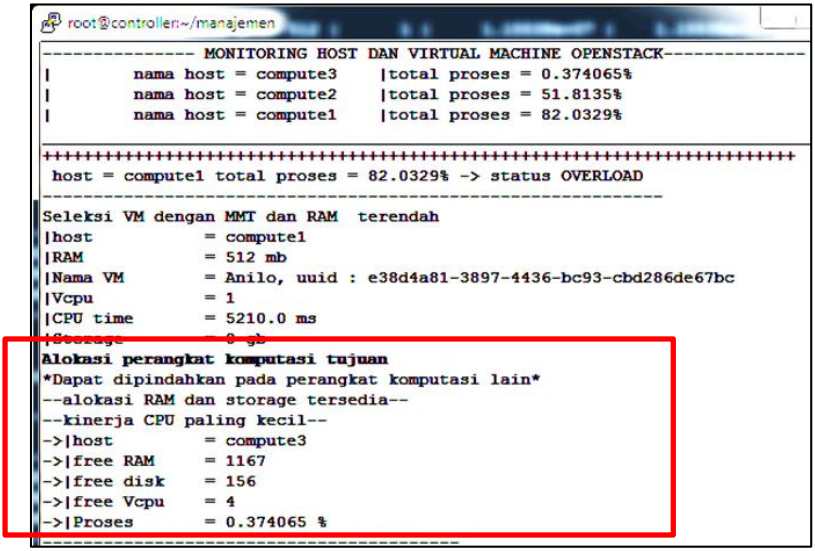


Gambar 4.20. Antarmuka Seleksi Mesin Virtual yang akan Dipindah

4.5.1.4. Tampilan Tujuan Perangkat yang Sesuai untuk Perpindahan Mesin Virtual

Pada Gambar 4.21 yang dikotak merah adalah tampilan mesin perangkat komputasi tujuan. Tampilan ini dapat ditampilkan ketika terdapat perangkat komputasi yang sesuai untuk perpindahan mesin virtual pada perangkat komputasi yang memiliki beban kerja berlebih. Pada tampilan ini terdapat nama perangkat komputasi

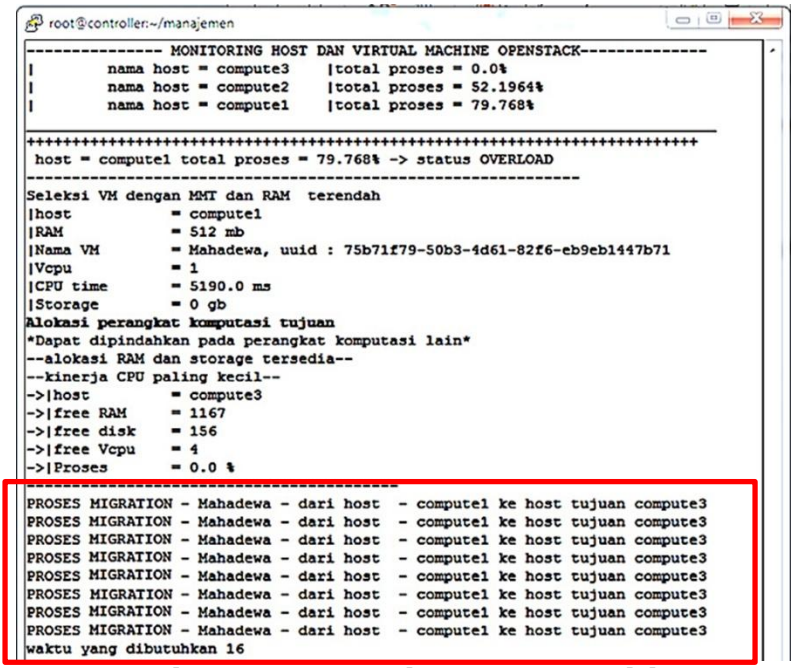
tujuan, nillai RAM yang tidak terpakai, nilai *storage* yang tidak terpakai, dan penggunaan CPU.



Gambar 4.21. Antarmuka Seleksi Perangkat Komputasi Tujuan

4.5.1.5. Antarmuka Proses Perpindahan Mesin Virtual Pada Perangkat Komputasi Tujuan

Pada Gambar 4.22 adalah tampilan proses perpindahan mesin virtual pada perangkat komputasi tujuan yang ditandai dengan kotak merah. Tampilan ini dapat ditampilkan ketika didapat nilai dari perangkat tujuan dari mesin virtual yang overload.

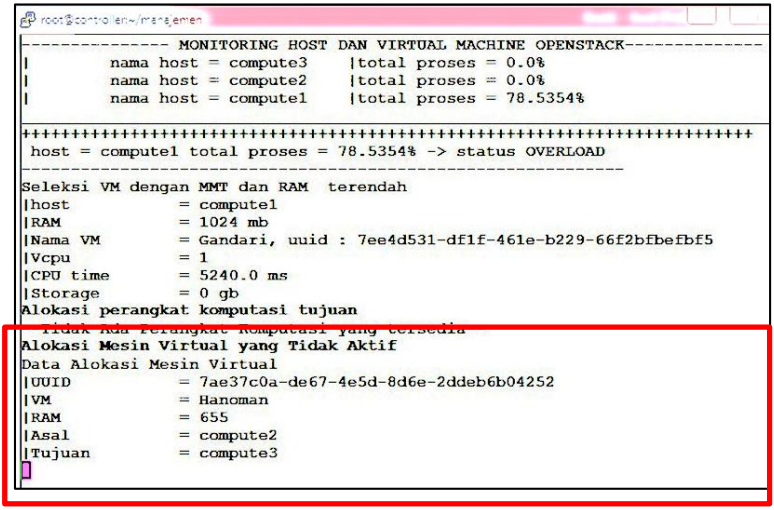


Gambar 4.22. Antarmuka Proses Perpindahan

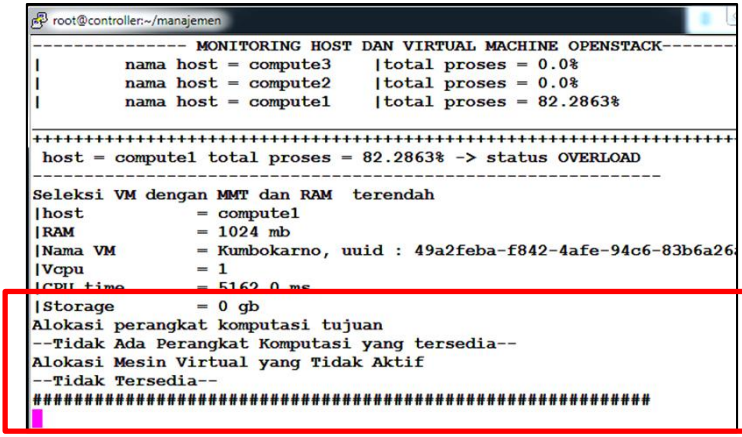
4.5.1.6.Tampilan Alokasi Mesin Virtual yang Tidak Aktif

Gambar 4.23 adalah tampilan proses alokasi mesin virtual yang tidak aktif untuk proses perpindahan mesin virtual pada perangkat komputasi yang mengalami beban kerja berlebih yang diperjelas dengan kotak merah. Tampilan ini berisi RAM, *Storage* dan tampilan ini dapat ditampilkan ketika tidak ada perangkat komputasi yang secara normal dapat dialokasikan untuk perpindahan mesin virtual pada perangkat komputasi yang mengalami beban kerja berlebih. Sedangkan pada Gambar 4.24 adalah tampilan dimana proses pengalokasian mesin virtual yang tidak aktif tetap tidak tersedia yang ditandai dengan kotak merah. Maka sistem akan menampilkan pesan bahwa perpindahan mesin

virtual tidak dapat dilakukan. Dan proses akan dilanjutkan pada tahap pertama kali sistem dimulai.



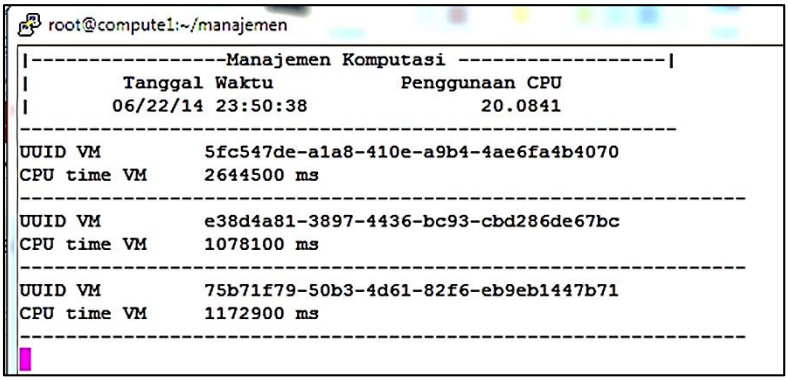
Gambar 4.23. Alokasi Mesin Virtual yang Tidak Aktif



Gambar 4.24. Tampilan Sistem Jika Alokasi Mesin Virtual yang Tidak Aktif Tidak tersedia

4.5.2. Tampilan Sistem *Cloud Child*

Tampilan utama dari Sistem Manajemen dapat dilihat pada Gambar 4.25, pada halaman ini pengguna dapat melihat data UUID pada mesin virtual penggunaan CPU yang ada pada perangkat.



Gambar 4.25. Tampilan Sistem *Cloud Child*

BAB V
UJI COBA DAN EVALUASI

Pada Bab ini akan membahas uji coba dan evaluasi dari sistem yang dibuat. Sistem akan diuji coba fungsionalitas dan performa dengan menjalakan skenario yang sudah ditentukan. Uji coba dilakukan untuk mengetahui hasil dari sistem ini sehingga dapat menjawab rumusan masalah pada Bab I.

5.1 Lingkungan Uji Coba

Pada subbab ini dijelaskan mengenai gambaran lingkungan yang digunakan untuk melakukan uji coba sistem. Uji coba sistem ini dilakukan dengan menggunakan satu buah komputer sebagai perangkat komputasi, tiga buah komputer sebagai perangkat komputasi, dan satu buah *switch* yang diletakkan pada satu ruangan. Spesifikasinya adalah sebagai berikut:

- Satu komputer rakitan.
  - Spesifikasi perangkat keras.
    - Intel (R) Core (TM) 2 Dou CPU E7400 @ 2.80GHz
    - 2 GB DDR2
    - 250 GB MAXTOR STM325031
    - RTL8111/8168/8411 PCI Express Gigabit Ethernet
  - Spesifikasi perangkat lunak:
    - CentOS 6.5 64 Bit sebagai sistem operasi, dengan *mode* instalasi minimal.
    - Glusterfs 3.5 sebagai penyedia layanan distribusi filesystem
    - Mysql 5.5.24 sebagai database yang digunakan
    - OpenStack sebagai *platform* penyedia komputasi awan.
- Dua komputer merk ACER.
  - Spesifikasi perangkat keras:
    - Intel (R) Core (TM) i3-2120 CPU @ 3.30GHz
    - 4 GB DDR3

  - 500GB WDC WD5000AAKX-2
  - Realtek RTL8111/8168B PCI Express Gigabit Ethernet
- Spesifikasi perangkat lunak.
  - CentOS 6.5 64 Bit sebagai sistem operasi, dengan *mode* instalasi minimal untuk satu komputer komputasi, dan *mode full* instalasi untuk satu komputer *gateway* dan komputasi.
  - Glusterfs 3.5 sebagai penyedia layanan distribusi filesystem
  - QEMU sebagai dasar virtualisasi pendukung.
  - KVM sebagai hypervisor didukung qemu dalam menjalankan mesin virtual.
  - Libvirt sebagai penyedia pustaka API dalam mengelola mesin virtual
  - OpenStack sebagai *platform* penyedia komputasi awan.

➢ Satu komputer HP.
- Spesifikasi perangkat keras:
  - Intel (R) Core (TM) i3-3220 CPU @ 3.30GHz
  - 2 GB DDR3
  - 500GB WDC WD5000AAKX-6
  - Realtek RTL8111/8168B PCI Express Gigabit Ethernet
- Spesifikasi perangkat lunak:
  - CentOS 6.5 64 Bit sebagai sistem operasi, dengan *mode* instalasi minimal.
  - Glusterfs 3.5 sebagai penyedia layanan distribusi filesystem
  - QEMU sebagai dasar virtualisasi pendukung.
  - KVM sebagai hypervisor didukung qemu dalam menjalankan mesin virtual.
  - Libvirt sebagai penyedia pustaka API dalam mengelola mesin virtual

- OpenStack sebagai *platform* penyedia infrastuktur komputasi awan.

➢ Satu buah switch D-link series DES-1210-28

Untuk lokasi uji coba perangkat lunak dilakukan di Laboratorium NCC Teknik Informatika. Sumber daya pembangunan komputasi awan menggunakan sumber daya komputer dari laboratorium dan menggunakan jaringan internet dari Institut Teknologi Sepuluh Nopember.

5.2 Skenario Uji Coba

Uji coba ini dilakukan untuk menguji apakah fungsionalitas yang diidentikasi pada tahap benar-benar diimplementasikan dan bekerja seperti seharusnya. Uji coba akan dilakukan dengan beberapa skenario untuk menguji kesesuaian respon kinerja sistem.

5.2.1 Uji Coba Fungsionalitas

Uji coba fungsionalitas merupakan sebuah pengujian yang dilakukan terhadap jalanya fungsi-fungsi utama pada sistem yang telah dibuat. Pengujian dilakukan ke seluruh fungsi baik itu di sisi *Cloud Child* maupun di *Cloud Head*. Uji coba fungsionalitas meliputi semua alur program yang sudah dijelaskan pada Bab III diantaranya sebagaiberikut.

a. Pengecekan Kinerja Perangkat Komputasi dan Mesin Virtual (UC-01)
   Pengecekan kinerja perangkat komputasi dan mesin virtual merupakan salah satu fungsi yang dilakukan untuk memperoleh data penggunaan CPU dan memperoleh data dari mesin virtual yang ada pada perangkat.
b. Menentukan Perangkat Komputasi yang Memiliki Beban Kerja Berlebih (UC-02)
   Setelah Perangkat komputasi mengirimkan data kinerja pada database *Cloud Head*, kemudian *Cloud Head* yang ada pada

kontroler mengolah data untuk mendapatkan data prangkat komputasi yang mengalami beban kerja berlebih.

c. Menentukan Mesin Virtual yang paling ideal untuk dipindah (UC-03).
   Dari data perangkat komputasi yang mengalai beban kerja berlebih, *Cloud Head* akan mengola data mesin virtual yang ada pada perangkat komputasi untuk mencari nilai dengan *Minimum Migration Time* dan beban CPU paling tinggi.
d. Menentukan Perangkat Komputasi Tujuan Perpindahan Mesin Virtual
   Setelah didapat data mesin virtual yang akan dipindah, *Cloud Head* akan mengalokasikan tujuan mesin virtual dengan mengolah data sumber daya perangkat komputasi lain atau *host* komputasi lain yang memiliki nilai ideal untuk proses perpindahan.
e. Alokasi Mesin Virtual dengan Status Tidak Aktif
   Proses alokasi mesin virtual dengan status tidak aktif bisa dilakukan jika dalam kondisi normal semua sumber daya semua perangkat tidak tersedia untuk proses perpindahan mesin virtual. *Cloud Head* akan mengalokasiakn mesin virtual dengan status tidak aktif dengan memindahkan ke perangkat lain yang sesuai, dengan ketentuan mesin virtual dan perangkat komputasi yang dialokasikan sesuai dengan kondisi mesin virtual yang dipindah.

5.2.2. Uji Coba Pengecekekan Kinerja Perangkat Komputasi dan Mesin Virtual

Uji coba ini dilakukan dengan menjalankan perangkat lunak *Cloud Child* di semua perangkat komputasi yang terhubung. Kemudian uji coba dilanjutkan dengan melakukan pengecekan kondisi perangkat komputasi untuk memperoleh data kinerja CPU dan mesin virtual yang ada pada perangkat secara *real time*. Data yang didapat kemudian disimpan ke dalam *database* terpusat yang ada pada perangkat kontroler.

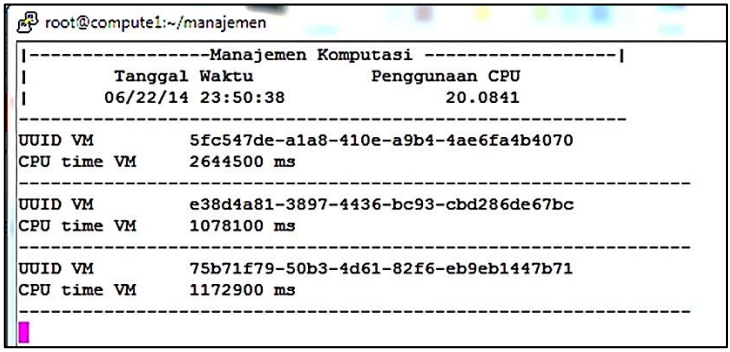
Tabel 5.1. Prosedur Uji Coba Pengecekan Perangkat Komputasi dan Mesin Virtual

ID	**UJ-01**
Nama	**Uji Coba Pengecekan Kinerja Perangkat Komputasi dan Mesin Virtual**
Tujuan Uji Coba	Menguji fitur sistem untuk melakukan pengecekan kinerja CPU dan mesin virtual yang ada pada perangkat komputasi
Kondisi Awal	Sistem *cloud child* belum dijalankan.
Skenario	**Menjalankan sistem *cloud child* untuk memulai pengecekan pada semua perangkat komputasi.**
Masukan	-
Keluaran	Data kinerja CPU dan mesin virtual yang ada pada perangkat.
Hasil uji coba	**Berhasil**

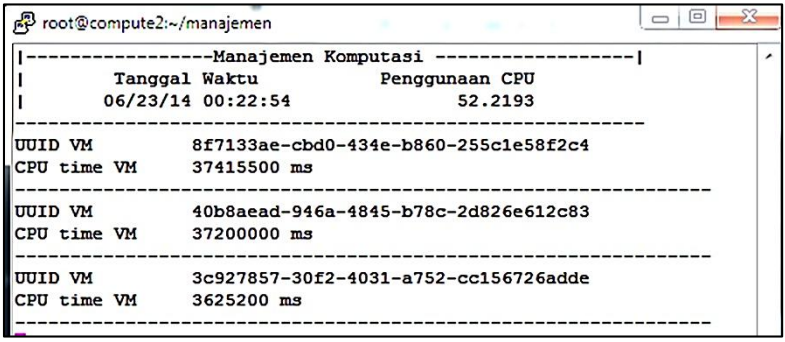
Sesuai yang dijelaskan pada Tabel 5.1, uji coba dilakukan dengan menjalankan sistem *Cloud Child* pada semua perangkat. Setelah sistem dijalankan, sistem akan mulai memindai kondisi penggunaan CPU dan mesin virtual yang ada pada perangkat. Gambar 5.1, Gambar 5.2, dan Gambar 5.3 merupakan tampilan sistem ketika melakukan pengecekan kinerja komputasi dan mesin virtual.

Data yang didapat dari pemindaian kemudian disimpan pada *database* pusat yang ada pada perangkat kontroler. Gambar 5.4. merupakan data dari penggunaan CPU pada semua perangkat komputasi yang berhasil disimpan dalam *database*. Sedangkan gambar Gambar 5.5. merupakan data semua mesin virtual yang berhasil disimpan dalam *database.* Uji coba dilakukan untuk

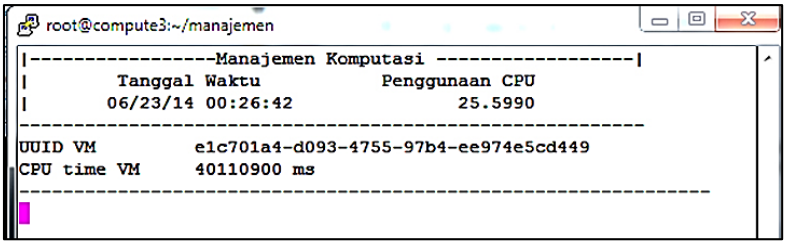
mendapatkan data dari semua perangkat komputasi atau *host* yang kemudian digunakan oleh *coud head* untuk monitoring keadaan peenggunaan CPU dan mesin virtual. Dari data tersebuat dapat dilakukan beberapa pengolahan seperti menentukan *host* yang mengalamai beban kerja berlebih.



Gambar 5.1. Uji Coba Ketika Menjalankan *Cloud child* Pada Compute1



Gambar 5.2. Uji Coba Ketika Menjalankan *Cloud child* Pada Compute2



Gambar 5.3. Uji Coba Ketika Menjalankan *Cloud child* Pada Compute3

host_id	host_name	host_cpu_usage
2	compute3	25.1889
3	compute2	51.8758
4	compute1	4.46781

Gambar 5.4. Output Data Penggunaan CPU dalam persentase perangkat komputasi pada *database*

vm_id	vm_uuid	vm_total_cpu
2745	e1c701a4-d093-4755-97b4-ee974e5cd449	30900
2743	40b8aead-946a-4845-b78c-2d826e612c83	31300
2744	3c927857-30f2-4031-a752-cc156726adde	1200
2739	5fc547de-a1a8-410e-a9b4-4ae6fa4b4070	3100
2740	e38d4a81-3897-4436-bc93-cbd286de67bc	3100
2741	75b71f79-50b3-4d61-82f6-eb9eb1447b71	2900
2742	8f7133ae-cbd0-434e-b860-255c1e58f2c4	31300

Gambar 5.5. Output Data Semua Mesin Virtual yang ada pada perangkat komputasi

5.2.3. Uji Coba Menentukan Perangkat Komputasi yang Memiliki Beban Kerja Berlebih

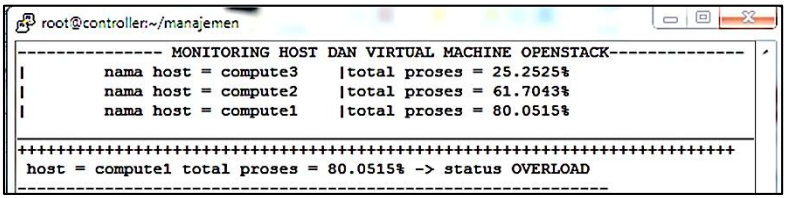
Uji coba ini dilakukan untuk mendapatkan nilai perangkat atau host komputasi yang menmiliki beban kerja berlebih. Uji coba dimulai dengan menjalankan sistem *Cloud Head* di perangkat

kontroler. Kemudian *Cloud Head* mengolah data penggunaan CPU perangkat komputasi yang ada pada *database* untuk menentukan kinerja perangkat komputasi yang berlebih.

Tabel 5.2. Prosedur Uji Coba Menentukan Perangkat Komputasi yang Memiliki Beban Kerja Berlebih

ID	**UJ-02**
Nama	**Uji Coba Menentukan Perangkat Komputasi yang Memiliki Beban Kerja Berlebhih**
Tujuan Uji Coba	Menguji fitur untuk monitoring kinerja perangkat komputasi dan menentukan perangkat komputasi yang memiliki memen kerja yang berlebih.
Kondisi Awal	Sistem *cloud head* belum dijalankan dan data penggunaan CPU sudah tersedia.
Skenario	**Menjalankan sistem *cloud head* kemudian sistem akan melakukan monitoring dan menunjukan perangkat komputasi yang memiliki beban kerja berlebih.**
Masukan	Data penggunaan CPU semua perangkat komputasi .
Keluaran	Data perangkat komputasi yang memiliki beban kerja berlebih.
Hasil uji coba	**Berhasil**

Sesuai yang dijelaskan pada Tabel 5.2, uji coba dilakukan dengan mendapatkan data hasil dari *Cloud Child* berupa penggunaan CPU. Kondisi awal data penggunaan CPU pada perangkat komputasi sudah tersedia tetapi sistem *Cloud Head* belum dijalankan. Keluaran yang diharapkan yaitu perangkat komputasi yang memiliki penggunaan CPU atau beban kerja berlebih.



Gambar 5.6. Uji Coba Perangkat Komputasi yang memiliki beban kerja Berlebih

Gambar 5.6. merupakan data perangkat komputasi yang mengalami penggunan CPU atau beban kerja berlebih yang tersimpan dalam *database*. Pada gambar diatas didapat data compute1 yang memiliki beban kerja 80% itu berarti compute1 memiliki beban kerja lebih karena mempunyai nilai melewati batas yaitu 75%. Sedangkan data compute2 adalah 65 % , dan compute3 hanya berkirsar 25%. Uji coba dilakukan untuk menentukan *host* mana yang memiliki beban kerja berlebih, yang nantinya proses perpindahan untuk membagi kinerja beban kerja CPU dapat dijalankan.

5.2.4. Uji Coba Seleksi Mesin Virtual Untuk Perpindahan

Uji coba dilakukan untuk mendapatkan mesin virtual yang mempunyai nilai untuk perpindahan paling ideal yang ada di dalam perangkat komputasi komputasi. Uji coba ini dilakukan dengan menampilkan mesin virtrual dengan nilai MMT paling rendah dan beban kerja CPU paling berat yang didapat dari selisih CPU *time* sebelum pengecekan dan sesuah pengecekan sistem manajemen sumber daya.

Tabel 5.3. Prosesudr Uji Coba Menentukan Mesin Virtual yang Ideal untuk Perpindahan

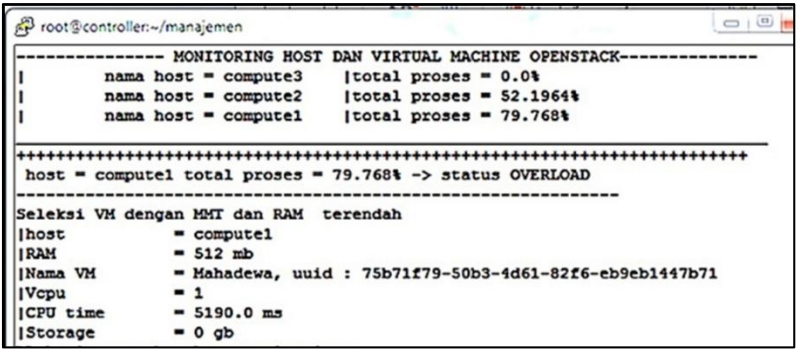
ID	**UJ-03**
Nama	**Uji Coba Menentukan Mesin Virtual yang Ideal Untuk Perpindahan**
Tujuan Uji Coba	Menguji fitur untuk menampilkan data mesin virtual pada perangkat komputasi dengan beban kerja berlebih yang mempunyai nilai ideal untuk proses perpindahan ke komputasi lain.
Kondisi Awal	Sistem *cloud head* telah dijalankan dan mendapatkan data perangkat komputasi yang mengalami beban kerja berlebih.
Skenario	**Mendapatkan data dan menampilkan mesin virtual yang ideal untuk proses perpindahan dari perangkat komputasi yang memiliki beban kerja berlebih.**
Masukan	Data perangkat komputasi yang memiliki beban kerja berlebih
Keluaran	Data mesin virtual yang memiliki nilai ideal untuk perpindahan.
Hasil uji coba	**Berhasil**

Berdasarkan prosedur pada Tabel 5.3. uji coba dilakukan dengan kondisi awal sudah mendapatkan data perangkat komputasi yang mengalami beban kerja berlebih. Kemudian setalah didapat data dari *database* seperti pada Tabel 5.4, maka sistem akan mencari nilai `memory_mb` sebagai nilai RAM dan `vm_total_CPU` sebagai CPU *time*. Nilai dari mesin virtual yang akan dipindah didapat dari nilai RAM paling kecil atau MMT paling minimum, jika tersedia RAM paling kecil maka sistem akan mencari nilai CPU time paling besar. Menentukan nilai RAM paling kecil adalah untuk menentukan MMT paling kecil.

Sedangkan CPU *time* paling besar untuk menentukan penggunaan CPU paling tinggi, sehingga ketika perpindahan alokasi akan lebih efisien dalam mengurangi beban kerja perangkat komputasi. Dari data Tabel 5.4 didapat data mesiin virtual dengan nama Anilo, RAM 512 MB yang mempunyai nilai RAM paling kecil dan CPU *time* 5040 dengan satuan *millisecond.*

Tabel 5.4. *Data Base* Mesin Virtual Pada Perangkat Komputasi

memory_mb	Host	display_name	vm_state	vm_CPU_time(Δt)
512	Compute1	Anilo	active	5040
512	Compute1	Mahadewa	active	2010
2048	Compute1	Werkudara	active	5040



Gambar 5.7. Hasil Mesin Virtual yang Akan Dipindah

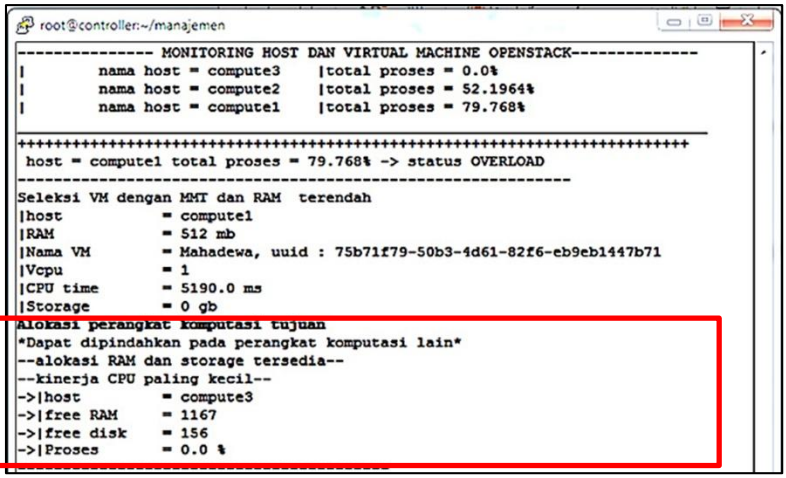
Pada Gambar 5.7 merupakan hasil dari data mesin virtual yang akan dipindah pada compute2. Data yang ditampilkan berupa RAM, nama mesin virtual, VCPU, CPU *time*, dan *storage.*

5.2.5. Uji Coba Menentukan Perangkat Komputasi Tujuan Untuk Perpindahan Mesin Virtual.

Uji coba ini dilakukan dengan menampilkan data perangkat komputasi dengan sumber daya yang sesuai untuk alokasi tujuan mesin virtual dengan nilai ideal yang akan dipindah.

Tabel 5.5. Prosedur Uji Coba Mendapatkan Perangkat Komputasi Tujuan Perpindahan Mesin Virtual

ID	**UJ-04**
Nama	**Uji Coba Mendapatkan Perangkat Komputasi Tujuan Perpindahan Mesin Virtual**
Tujuan Uji Coba	Menguji fitur untuk mendapatkan tujuan perangkat komputasi dengan sumber daya yang tersedia dan melakukan perpindahan ke perangkat komputasi tujuan.
Kondisi Awal	Telah memperoleh data mesin virtual yang akan dipipndah dan tersedianya sumber daya perangkat komputasi (RAM dan storage) dengan mesin virtual perpindahan.
Skenario	**Mendapatkan nilai mesin virtual yang dipindah, kemudian menentukan tujuan untuk proses perpindahan dengan mengalokasikan sumber daya preangkat komputasi yang tersedia**
Masukan	Data mesin virtual yang akan dipindah
Keluaran	Perpindahan mesin virtual
Hasil uji coba	**Berhasil**



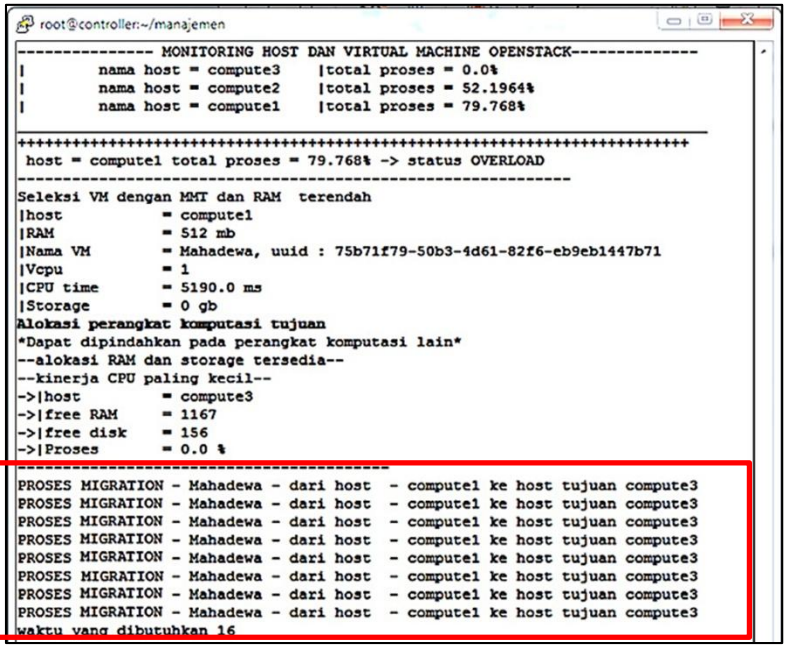
Gambar 5.8. Hasil Uji Coba Menentukan Perangkat Komputasi Tujuan

Sesuai dengan prosedur pada tabel Tabel 5.5, skenario digunakan untuk mendapatkan data tujuan perpindahan dengan sumber daya yang tersedia pada perangkat komputasi. Uji coba dilakukan dengan mengalokasikan sumber daya (RAM dan *storage*) yang tersedia pada perangkat komputasi yang sesui dengan mesin virtual perpindahan. Setelah didapat nilai alokasi sumber daya yang sesuai, sistem akan memilih perangkat komputasi yang memiliki beban kerja paling sedikit.

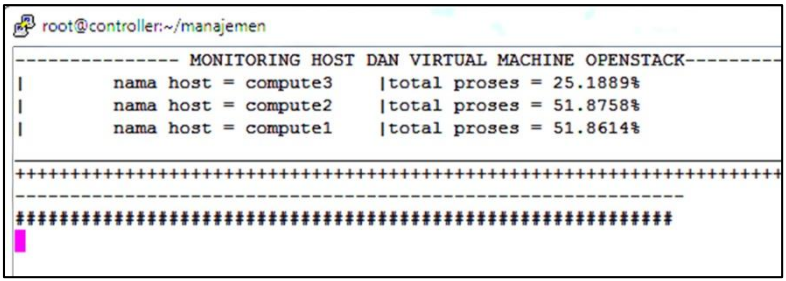
Hasil uji coba mendapatkan tujuan perpindahan dapat dilihat pada kotak merah pada Gambar 5.8. Kemudian sistem *cloud head* akan melakukan perpindahan mesin virtual ke perangkat komputasi tujuan dengan *live-migration*.

Proses live-migration dapat dilihat pada Gambar 5.9 yang diperjelas dengan kotak warna merah. Setelah proses *live-migration* selesai muncul data berupa lama proses *live-migration* ditunjukan pada Gambar 5.9 dengan nilai 16 detik. Setelah proses *live-migration* selesai, dapat dilihat pada Gambar 5.10. bahwa proses *live-migration* berhasil dengan berpindahanya mesin virtual pada perangkat berlebih ke perangkat tujuan yang memiliki beban

CPU rendah yaitu pada perangkat compute3. Berhasilnya perpindahan ditandai dengan tidak ada perangkat komputasi yang memiliki beban kerja berlebih yaitu compute1 dengan nilai penggunaan CPU 79%, menjadi 51 % dan beban kerja yang mempunyai kinerja kecil akan mengalami peningkatan penggunaan CPU yaitu compute3 dari 0% menjadi 25%. Perubahan sumber daya perangkat ketika sebelum dan sesudah proses *live migration* dapat dilihat pada Tabel 5.6. Untuk tampilan *dashboard* OpenStack sebelum perpindahan dapat dilihat pada Gambar 5.11 dan setelah proses perpindahan dapat dilihat pada Gambar 5.12 yang ditegaskan dengan kotak warna merah.



Gambar 5.9. Proses *Live-Migration* Mesin Virtual.



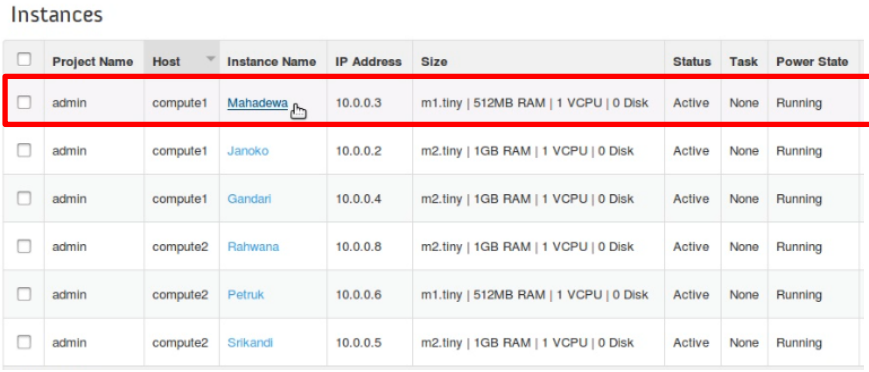
Gambar 5.10. Tampilan Cloud Head Setelah Proses *Live-Migration*

Tabel 5.6. Perubahan Data Setelah Proses *Live-Migration*

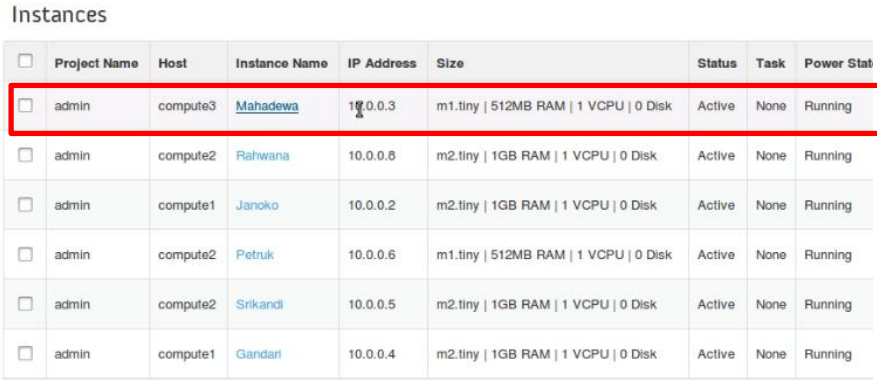
Time	Host	*Free RAM (MB)*	Proses CPU (%)	Overload	Nama Mesin Virtual	RAM (MB)	Status
1	Compute1	674	79.76	YA	Mahadewa	512	Aktif
					Janoko	1024	Aktif
					Gandari	1024	Aktif
					Werkudara	512	Aktif
	Compute2	609	52.19	Tidak	Petruk	512	Aktif
					Srikandi	1024	Aktif
					Rahwana	1024	Aktif
	Compute3	1679	0	Tidak	***Tidak ada***	***Tidak Ada***	***Tidak Ada***
2	Compute1	1189	51.86	Tidak	Janoko	1024	Aktif
					Gandari	1024	Aktif
					Werkudara	512	Aktif
	Compute2	609	52.19	Tidak	Petruk	1024	Aktif
					Srikandi	1024	Aktif
					Rahwana	1024	Aktif
	Compute3	116	25.18	Tidak	Mahadewa	512	Aktif

Keterangan

Warna	Penjelasan
	Kondisi Perangkat yang mengalami beban kerja berlebih
	Mesin Virtual yang mempunyai nilai MMT paling minimum dan CPU *time* paling tinggi untuk proses perpindahan.



Gambar 5.11. Tampilan OpenStack *Dashboard* Sebelum Perpindahan



Gambar 5.12. Tampilan OpenStack *Dashboard* Sesudah Perpindahan

5.2.6. Alokasi Mesin Virtual Yang Tidak Aktif

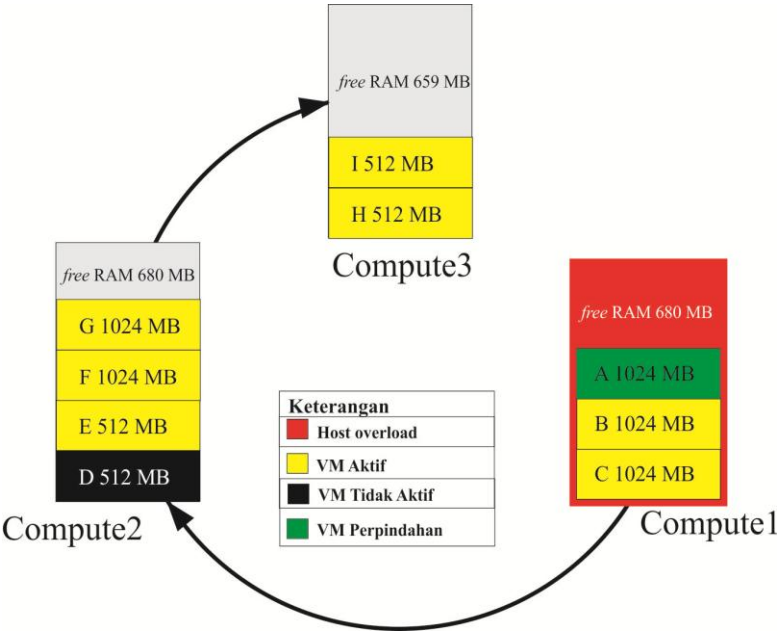
Uji coba ini diperlukan untuk membuktikan bahwa dengan alokasi mesin virtual yang tidak aktif dapat digunakan untuk menangani kasus jika dalam kondisi normal tidak tersedia sumber *daya* host untuk proses perpidahan mesin virtual pada *host* yang

mengalami beban kerja berlebih. Uji coba ini dilakukan dengan menampilkan data mesin virtual dengan status tidak aktif. Kemudian mengalokasikan perangkat komputasi yang sesuai untuk mesin virtual pada perangkat yang mengalami beban kerja komputasi berlebih dengan mesin virtual yang tidak aktif.

Tabel 5.7. Prosedur Uji Coba Alokasi Mesin Virtual yang Tidak Aktif.

ID	**UJ-05**
Nama	**Uji Coba Alokasi Mesin Virtual yang Tidak Aktif**
Tujuan Uji Coba	Menguji fitur untuk mengalokasikan mesin virtual yang tidak aktif untuk perpindahan mesin virtual pada perangkat dengan beban kerja berlebih.
Kondisi Awal	Tidak tersedia perangkat komputasi tujuan untuk proses perpindahan
Skenario	**Sistem akan memindai mesin virtual kondisi tidak aktif, kemudian mengalokasikan sumber daya komputasi lain untuk prosesn perpindahan mesin virtual.**
Masukan	Data mesin virtual yang akan dipindah
Keluaran	Alokasi sumber daya baru yang sesuai untuk perpindahan mesin virtual.
Hasil uji coba	**Berhasil.**

Sesuai dengan prosesdur pada Tabel 5.7, pengujian dapat dilakukan ketika di suatu perangkat komputasi mengalami kelebihan beban kerja tetapi tidak tersedia tujuan perpindahan untuk mengurangi beban kerja. Sehingga proses perpindahan mesin virtual tidak dapat dilakukan.

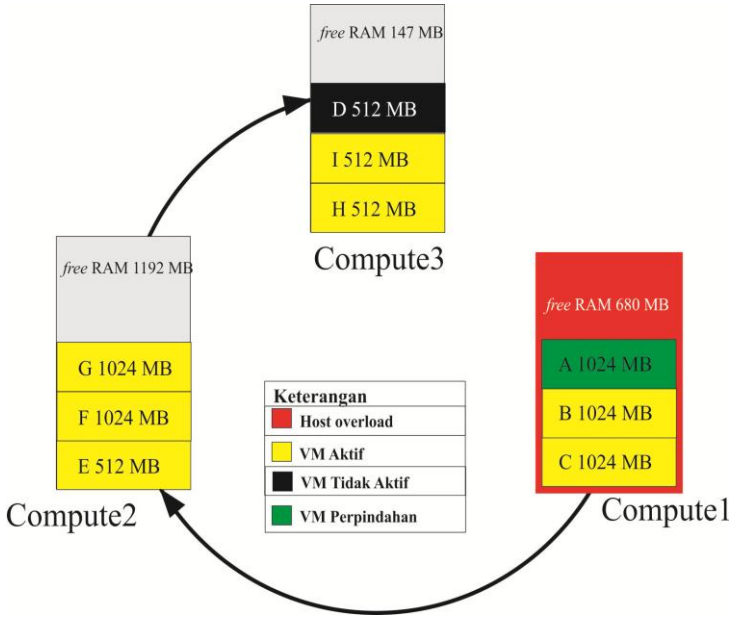


Gambar 5.13. Ilustrasi Uji Coba Alokasi Sumber Daya Mesin Virtual yang Tidak Aktif (1)

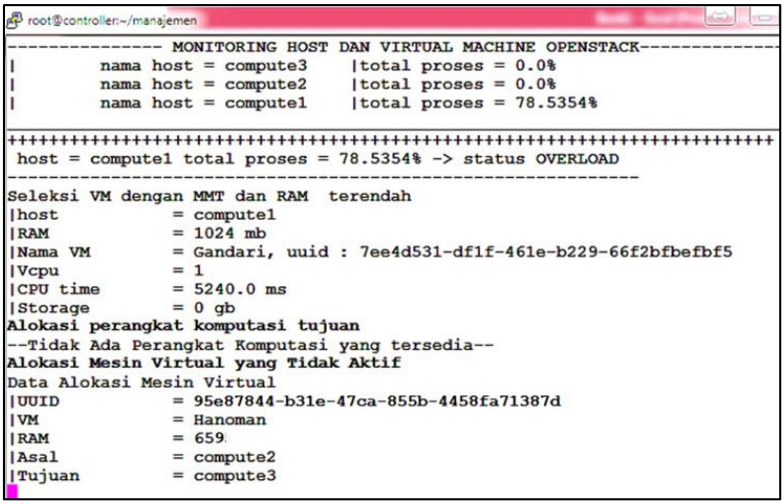
Seperti pada kasus diatas diperlukan alokasi dari mesin virtual yang tidak aktif untuk ditukar dengan mesin virtual pada perangkat komputasi dengan beban kerja berlebih. Untuk lebih jelasnya kasus uji coba dapat dilihat pada Gambar 5.13.

Dari Gambar 5.13. menjelaskan kondisi dimana alokasi mesin virtual yang tidak aktif dapat dilakukan. Tersedianya alokasi RAM pada perangkat lain seperti compute2 yang hanya mempunyai *free* RAM 680 MB dan compute3 hanya mempunyai *free* RAM 659 MB. Sehingga alokasi kedua perangkat komputasi tersebut tidak dapat digunakan untuk perpindahan mesin virtual pada perangakat yang mengalami beban kerja komputasi berlebih yaitu compute1 dengan nilai RAM 1024. Untuk itu sistem memerlukan alokasi mesin virtual yang tidak aktif supaya dapat

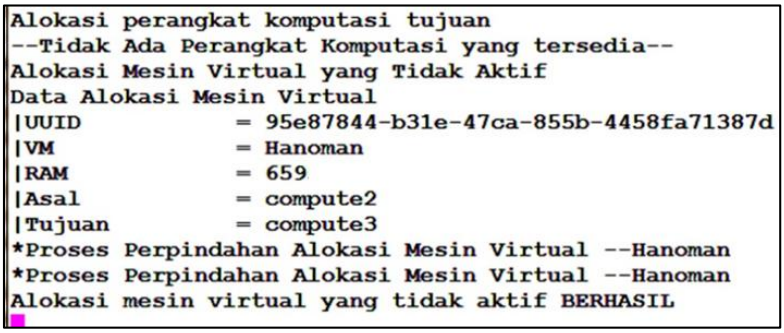
digunakan untuk pengurangan beban kerja berlebih. Ilustrasi kasus dapat dilihat pada Gambar 5.14, yaitu sistem melakukan perpindah mesin virtual yang tidak aktif pada gambar ditunjukan dengan warna hitam. Posisi semula mesin virtual yang tidak aktif pada compute2 kemudian dipindahkan ke compute3 seperti pada Gambar 5.14. Alokasi RAM dari compute2 menjadi 1192 MB yang semula 680 MB, sedangkan compute3 menjadi 147 MB yang semula 659 MB. Sehingga dengan alokasi RAM compute2 dapat digunkan untuk perpindahan mesin virtual pada *host* dengan beban kerja komputasi berlebih. Sedangkan tampilan pengalokasian mesin virtual yang tidak aktif pada sistem dapat dilihat pada Gambar 5.15. Proses perpindahan pada sistem untuk alokasi mesin virtual yang tidak aktif dapat dilihat pada Gambar 5.16.



Gambar 5.14. Ilustrasi Perpindhan Alokasi Mesin Virtual yang Tidak Aktif (2)



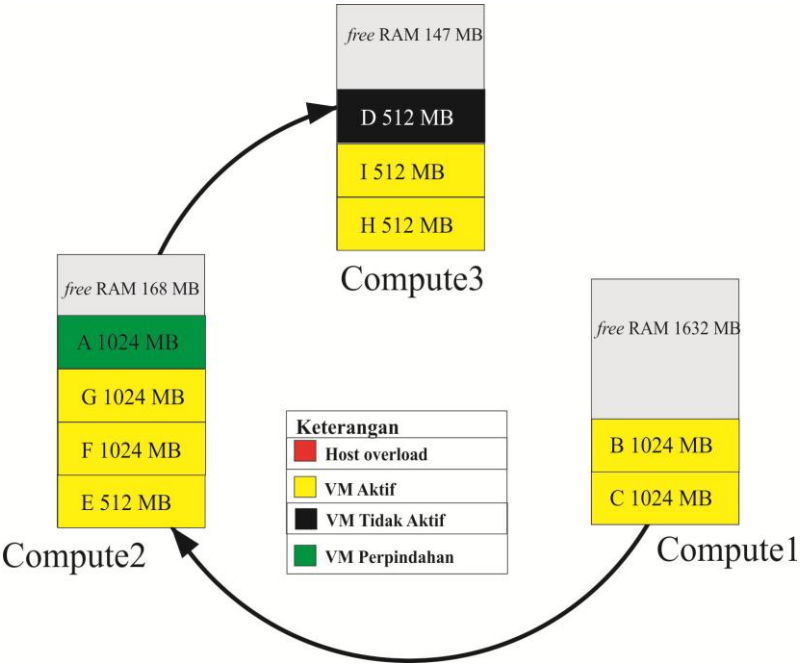
Gambar 5.15. Tampilan Alokasi Mesin Virtual yang Tidak Aktif



Gambar 5.16. Tampilan Perpindahan Mesin Virtual yang Tidak Aktif

Proses selanjutnya dapat diilustrasikan pada Gambar 5.17, mesin virtual yang tidak aktif telah dipindahkan ke compute3. Sehingga alokasi RAM pada compute2 dapat digunakan untuk perpindahan dengan nilai mesin virtual pada *host* dengan beban kerja komputasiberlebih adalah 1024 MB. Pada Gambar 5.17 merupakan tampilan perpindahan mesin virtual yang tidak aktif.

Setelah proses perpindahan alokasi RAM compute2 menjadi 97 MB dikarenakan sudah terisi mesin virtual dari *host* yang *overload.*



Gambar 5.17. Ilustrasi Akhir Proses Alokasi Mesin Virtual yang Tidak Aktif (3)

Proses selanjutnya adalah alokasi sumber daya perangkat komputasi dari mesin virtual yang tidak aktif kemudian digunakan mesin virtual pada pada perangkat yang *overload* untuk melakukan *live-migration*. Pada Gambar 5.18 merupakan tampilan perpindahan mesin virtual dari perangkat komputasi dengan beban kerja berlebih yang didapat dari alokasi mesin virtual yang tidak aktif. Perubahan sumber daya semua perangkat lebih jelasnya

dapat dilihat pada Tabel 5.8. Kondisi beban kerja CPU pada host dapat dilihat pada Gambar 5.19.

```
root@controller:~/manajemen
--------------- MONITORING HOST DAN VIRTUAL MACHINE OPENSTACK--------------
|         nama host = compute3     |total proses = 0.25413%
|         nama host = compute2     |total proses = 0.0%
|         nama host = compute1     |total proses = 80.0%

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 host = compute1 total proses = 80.0% -> status OVERLOAD
-------------------------------------------------------------
Seleksi VM dengan MMT dan RAM  terendah
|host           = compute1
|RAM            = 1024 mb
|Nama VM        = Gandari, uuid : 7ee4d531-df1f-461e-b229-66f2bfbefbf5
|Vcpu           = 1
|CPU time       = 5230.0 ms
|Storage        = 0 gb
Alokasi perangkat komputasi tujuan
*Dapat dipindahkan pada perangkat komputasi lain*
--alokasi RAM dan storage tersedia--
--kinerja CPU paling kecil--
->|host         = compute2
->|free RAM     = 1121
->|free disk    = 156
->|Proses       = 0.0 %
-----------------------------------------
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
PROSES MIGARTION - Gandari - dari host  - compute1 ke host tujuan compute2
waktu yang dibutuhkan 18
############################################################
Migration berhasil dilakukan dengan uuid compute1 ke host compute2 dari
dengan waktu 18 detik
```

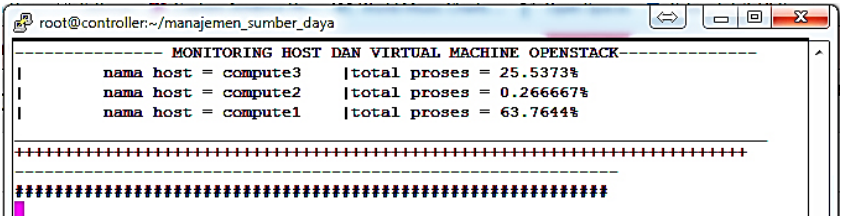
Gambar 5.18. Tampilan Perpindahan Mesin Virtual Pada Perangkat Overload Dari Alokasi Mesin Virtual yang Tidak Aktif

Tabel 5.8. Data Perubahan Sumber Daya Perangakt Komputasi dan Mesin Virtual

Time	Host	*Free RAM (MB) / Storage (GB)*	Proses CPU (%)	Overload	Nama Mesin Virtual	RAM (MB) / Storage (GB)	Status
1	Compute1	680/136	81.84	Ya	Gondari	1024/0	Aktif
					Janoko	1024/10	Aktif
					Kumbokarno	1024/10	Aktif
	Compute2	680/156	0.09	Tidak	Hanoman	512/0	Tidak
					Srikandi	1024/0	Tidak
					Rahwana	1024/0	Aktif
					Werkudara	512/0	Aktif
	Compute3	659/146	0.16	Tidak	Anilo	1024/20	Aktif
2	Compute1	162/156	81.84	Ya	Gondari	1024/0	Aktif
					Janoko	1024/10	Aktif
					Gondari	1024/10	Aktif
	Compute2	1192/156	0.09	Tidak	Srikandi	1024/0	Tidak
					Rahwana	1024/0	Aktif
	Compute3	147/156	0.3	Tidak	Anilo	1024/20	Aktif
					Hanoman	512/0	Tidak
3	Compute1	1186/156	56.84	Tidak	Janoko	1024/10	Aktif
					Gandari	1024/10	Aktif
	Compute2	97/156	0.09	Tidak	Gondari	1024/0	Aktif
					Srikandi	1024/0	Tidak
					Rahwana	1024/0	Aktif
	Compute3	143/156	25.16	Tidak	Anilo	5102420	Aktif
					Hanoman	512/0	Tidak

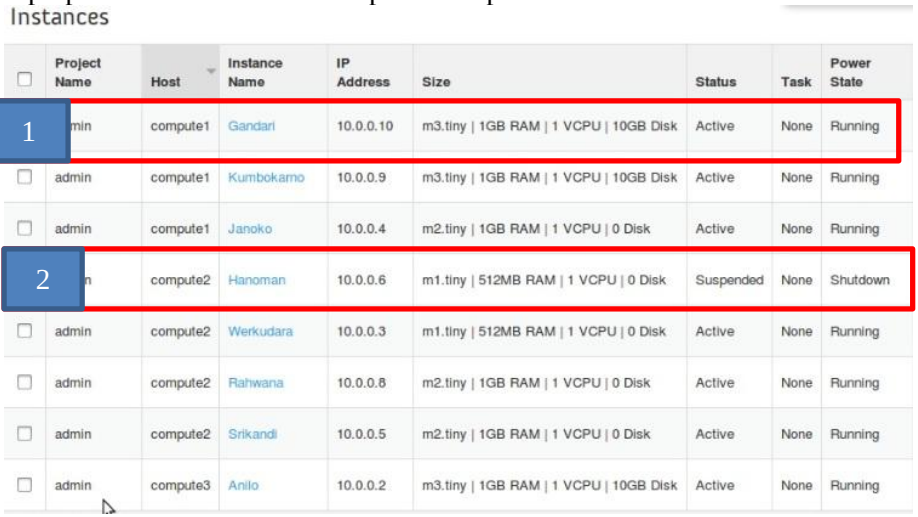
Keterangan

Warna	Penjelasan
	Kondisi Perangkat yang mengalami beban kerja berlebih
	Mesin virtual tidak aktif yang digunakan alokasi untuk *live migration*
	Mesin Virtual yang mempunyai nilai MMT paling minimum dan CPU *time* paling tinggi dan sebagai mesin virtual perpindahan.

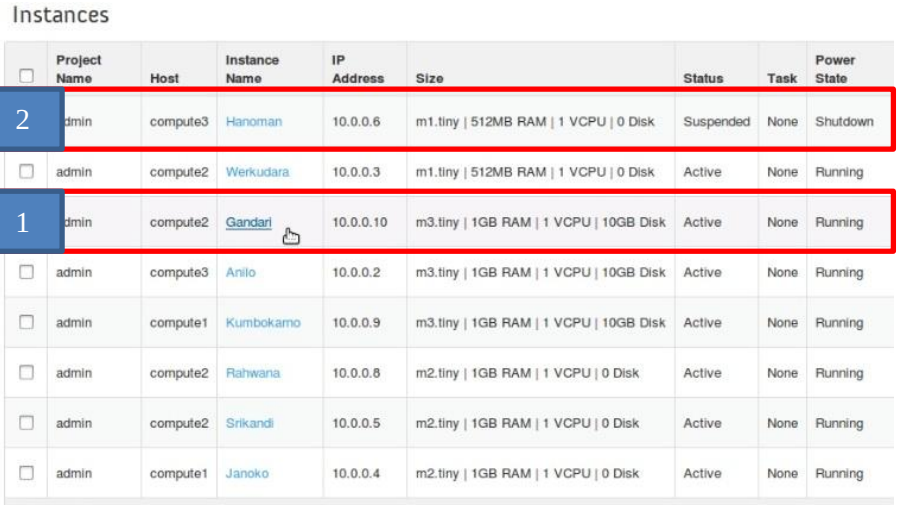


Gambar 5.19. Kondisi Beban Kerja *Host* Setelah Alokasi Mesin Virtual Tidak Aktf Untuk Perpindahan

Adapun tampilan pada OpenStack *dashboard* sebelum proses perpindahan mesin virtual dapat dilihat Gambar 5.20 dan sesudah perpindahan mesin virtual dapat diihat pada Gambar 5.21.



Gambar 5.20. Tampilan OpenStack *Dashboard* Sebelum Perpindahan Untuk Alokasi VM Tidak Aktif



Gambar 5.21. Tampilan OpenStack *Dashboard* Setelah Proses Perindahan Untuk Alokasi VM Tidak Aktif

Kotak merah pada kedua gambar untuk menunjukan mesin virtual yang mengalami perpindahan yang bertanda no 1 adalah mesin virtual pada beban kerja berlebih yang semula pada *host* Compute1 berubah menjadi *host* Compute2, sedangkan no 2 adalah mesin virtual yang tidak aktif yang semula berada di *host* Compute2 berubah menjadi *host* Compute3.

5.2.2 Uji Coba Performa

Pada bagian ini dilakukan uji coba performa untuk mengatahui perilaku dari sistem ketika dijalankan pada keadaan sebenarnya. Uji coba ini dilakukan utntuk mengetahui efisensi kerja sistem sebagai pengatur pembagian kinerja *host*, dan mengetahui berapa lama perubahan kinerja *host* menjadi stabil dan tidak mengalami kelebihan beban kerja komputasi setelah melakukan perpindahan.

5.2.2.1 Efisiensi Pembagian Beban Kerja Perangkat Komputasi

Pada Tugas Akhir ini efisensi beban kerja dihitung dari data penggunaan CPU secara *real time*. Uji coba dimulai dengan mengambil data dari penggunaan beban kerja CPU sebelum sistem dijalankan, data dapat dilihat pada Tabel 5.9 untuk beban kerja CPU pada *host* compute1, Tabel 5.10 untuk beban kerja pada *host* compute2, Tabel 5.11 untuk beban kerja pada host compute3. Uji coba dilakukan dengan memberikan beban kerja terus menerus sampai *host* komputasi mengalami kelebihan beban kerja.

Tabel 5.9 Beban Kerja CPU Pada *Host* Compute1Tanpa Sistem Manajemen Sumber Daya

State	*Host*	Proses *CPU (%)*	*Overload*	Tambah Beban Kerja
1	Compute1	0.2		Ya
	Compute2	0.1		
	Compute3	0.3		
2	Compute1	26.1		Ya
	Compute2	0.5		
	Compute3	0.4		
3	Compute1	51.6		
	Compute2	0.1		
	Compute3	0.8		
4	Compute1	76.1	Ya	
	Compute2	0.2		
	Compute3	0.4		
5	Compute1	80.1	Ya	
	Compute2	0.4		
	Compute3	0.4		
6	Compute1	98.0	Ya	
	Compute2	0.2		
	Compute3	0.4		
7	Compute1	98.2	Ya	
	Compute2	2.2		
	Compute3	3.8		
8	Compute1	80.2	Ya	
	Compute2	0.2		
	Compute3	0.9		

State	*Host*	Proses *CPU (%)*	*Overload*	Tambah Beban Kerja
9	Compute1	90.2	Ya	
	Compute2	0.9		
	Compute3	0.4		
10	Compute1	86.5	Ya	
	Compute2	0.5		
	Compute3	0.3		
11	Compute1	86.2	Ya	
	Compute2	0.9		
	Compute3	0.3		
12	Compute1	87.3	Ya	
	Compute2	0.2		
	Compute3	0.8		
13	Compute1	95	Ya	
	Compute2	0.2		
	Compute3	0.7		

Keterangan

	Penambahan beban kerja
	Host overload dan tidak tersedia *live migration*

Uji coba dilanjutkan dengan menghitung beban kerja *host* compute2 sebelum sistem dijalankan.

Tabel 5.10 Beban Kerja CPU Pada *Host* Compute2 Tanpa Sistem Manajemen Sumber Daya

State	*Host*	Proses *CPU (%)*	*Overload*	Tambah Beban Kerja
1	Compute1	1.2		
	Compute2	0.1		Ya
	Compute3	0.3		
2	Compute1	3.1		
	Compute2	27.0		Ya
	Compute3	0.4		
3	Compute1	2.3		
	Compute2	58.0		Ya
	Compute3	0.8		
4	Compute1	3.2		
	Compute2	76.0	Ya	
	Compute3	0.4		
5	Compute1	0.1		

State	*Host*	Proses *CPU (%)*	*Overload*	Tambah Beban Kerja
	Compute2	79.4	Ya	
	Compute3	5.4		
6	Compute1	5.2		
	Compute2	80.2	Ya	
	Compute3	0.4		
7	Compute1	2.2		
	Compute2	75.2	Ya	
	Compute3	2.3		
8	Compute1	3.2		
	Compute2	76.3	Ya	
	Compute3	0.4		
9	Compute1	3.2		
	Compute2	76.9	Ya	
	Compute3	3.4		
10	Compute1	5.5		
	Compute2	75.5	Ya	
	Compute3	1.3		
11	Compute1	10.2		
	Compute2	76.9	Ya	
	Compute3	0.3		
12	Compute1	5.3		
	Compute2	75.2	Ya	
	Compute3	0.8		
13	Compute1	7.3		
	Compute2	78.2	Ya	
	Compute3	0.7		

Uji coba dilanjutkan dengan menghitung beban kerja *host* pada compute3 sebelum sistem dijalankan.

Tabel 5.11 Beban Kerja CPU Pada *Host* Compute3 Tanpa Sistem Manajemen Sumber Daya

State	*Host*	Proses *CPU (%)*	*Overload*	Tambah Beban Kerja
1	Compute1	3.2		
	Compute2	0.1		
	Compute3	0.3		Ya
2	Compute1	2.1		
	Compute2	0.5		
	Compute3	25.0		Ya

State	*Host*	Proses *CPU (%)*	*Overload*	Tambah Beban Kerja
3	Compute1	4.6		
	Compute2	0.1		
	Compute3	55.1		Ya
4	Compute1	5.3		
	Compute2	0.4		
	Compute3	78.0	Ya	
5	Compute1	5.1		
	Compute2	0.4		
	Compute3	79..4	Ya	
6	Compute1	9.2		
	Compute2	0.2		
	Compute3	76.4	Ya	
7	Compute1	3.1		
	Compute2	0.7		
	Compute3	78.8	Ya	
8	Compute1	4.2		
	Compute2	0.3		
	Compute3	75.4	Ya	
9	Compute1	2.3		
	Compute2	0.4		
	Compute3	75.4	Ya	
10	Compute1	5.5		
	Compute2	0.5		
	Compute3	79.3	Ya	
11	Compute1	5.2		
	Compute2	0.9		
	Compute3	78.3	Ya	
12	Compute1	3.3		
	Compute2	0.2		
	Compute3	79.8	Ya	
13	Compute1	5.3		
	Compute2	0.2		
	Compute3	76.7	Ya	

Dikarenakan beban kerja CPU pada kondisi seperti Tabel 5.9, Tabel 5.10, dan Tabel 5.11 tidak maksimal, ketika satu *host* diberikan beban kerja berlebih tetapi *host* lain yang sedang dalam keadaan komputasi rendah. *Host* dengan kinerja rendah seharusnya dapat digunakan untuk membantu kinerja *host* yang overload, maka sistem seperti ini kurang efisien dalam melakukan proses komputasi pada komputasi awan. Untuk itu diperlukan uji coba

efisensi pembagian beban kerja perangkat komputasi untuk membuktikan pembagian kerja yang lebih efiktif. Uji coba dimulai ketika semua *host* dalam keadaan tidak mendapat beban kerja dari mesin virtual. Kemudian uji coba dilanjutkan dengan memberikan beban kerja pada semua *host.* Penambahan beban kerja dapat dilakukan dengan memberikan beban kerja tambahan pada mesin virtual yang ada pada *host* sehingga beban kerja pada *host* akan meningkat, begitu seterusnya sampai tidak ada mesin virtual lagi yang memungkinkan untuk proses perpindahan yang bertujuan mengurangi beban kerja *host.*

Data pertambahan beban kerja pada *host* dapat dilihat pada pada Tabel 5.12 untuk *host* compute1, Tabel 5.13 untuk *host* compute2, dan Tabel 5.14 compute3. Ketiga tebel tersebut merupakan data dari kondisi beban kerja CPU setelah sistem dijalankan.

Tabel 5.12. Data Efisiensi Kinerja CPU Pada *Host* Compute1

State	*Host*	Proses *CPU (%)*	*Overload*	*Live Migration*	Tambah Beban Kerja
1	Compute1	0.2			Ya
	Compute2	0.1			
	Compute3	0.3			
2	Compute1	26.1			Ya
	Compute2	0.5			
	Compute3	0.4			
3	Compute1	51.6			
	Compute2	0.1			
	Compute3	0.8			
4	Compute1	76.1	Ya	Ya	
	Compute2	0.2			
	Compute3	0.4			
5	Compute1	56.1			Ya
	Compute2	25.4			
	Compute3	0.4			
6	Compute1	82.2	Ya	Ya	
	Compute2	26.2			
	Compute3	0.4			
7	Compute1	52.2			Ya

State	*Host*	Proses *CPU (%)*	*Overload*	*Live Migration*	Tambah Beban Kerja
	Compute2	27.2			
	Compute3	28.8			
8	Compute1	79.2	Ya	Ya	
	Compute2	26.2			
	Compute3	25.4			
9	Compute1	56.2			Ya
	Compute2	26.9			
	Compute3	51.4			
10	Compute1	86.5	Ya	Ya	
	Compute2	29.5			
	Compute3	59.3			
11	Compute1	52.2			Ya
	Compute2	56.9			
	Compute3	58.3			
12	Compute1	80.3	Ya		
	Compute2	60.2			
	Compute3	59.8			
13	Compute1	95	Ya		
	Compute2	60.2			
	Compute3	56.7			

Keterangan

	Penambahan beban kerja
	Host overload dan tersedia *live migration*
	Host overload dan tidak tersedia *live migration*

Tabel 5.13. Data Efisiensi Kinerja CPU Pada *Host* Compute2

State	*Host*	Proses *CPU (%)*	*Overload*	*Live Migration*	Tambah Beban Kerja
1	Compute1	0.2			
	Compute2	0.1			
	Compute3	0.3			
2	Compute1	0.5			
	Compute2	26.1			
	Compute3	0.4			
3	Compute1	0.2			
	Compute2	53.6			
	Compute3	0.8			
4	Compute1	3.0			
	Compute2	76.1			

	Compute3	0.2			
5	Compute1	25.4			
	Compute2	56.1			
	Compute3	0.4			
6	Compute1	30.1			
	Compute2	82.4			
	Compute3	0.4			
7	Compute1	34.7			
	Compute2	54.2			
	Compute3	28.8			
8	Compute1	31.1			
	Compute2	79.1			
	Compute3	25.4			
9	Compute1	29.2			
	Compute2	51.9			
	Compute3	51.4			
10	Compute1	29.5			
	Compute2	86.2			
	Compute3	59.3			
11	Compute1	51.2			
	Compute2	56.9			
	Compute3	60.3			
12	Compute1	57.3			
	Compute2	82.2			
	Compute3	59.8			
13	Compute1	60.1			
	Compute2	95.2			
	Compute3	50.7			

Tabel 5.14. Data Efisiensi Kinerja CPU Pada *Host* Compute3

State	*Host*	Proses *CPU (%)*	*Overload*	*Live Migration*	Tambah Beban Kerja
1	Compute1	0.2			
	Compute2	1.3			
	Compute3	0.2			
2	Compute1	0.5			
	Compute2	0.4			
	Compute3	26.1			Ya
3	Compute1	0.2			
	Compute2	3.1			
	Compute3	53.6			Ya
4	Compute1	1.5			
	Compute2	4.1			

	Compute3	76.1	Ya	Ya	
5	Compute1	25.4			
	Compute2	0.3			
	Compute3	56.1			
6	Compute1	28.1			
	Compute2	2.1			
	Compute3	80.4	Ya	Ya	
7	Compute1	25.2			
	Compute2	26.7			
	Compute3	54.2			Ya
8	Compute1	31.1			
	Compute2	25.4			
	Compute3	79.1	Ya	Ya	
9	Compute1	26.2			
	Compute2	51.9			
	Compute3	51.4			Ya
10	Compute1	29.5			
	Compute2	51.5			
	Compute3	79.5	Ya	Ya	
11	Compute1	51.2			
	Compute2	56.9			
	Compute3	60.3			Ya
12	Compute1	50.3			
	Compute2	52.2			
	Compute3	82.8	Ya		
13	Compute1	60.1			
	Compute2	64.1			
	Compute3	98.0	Ya		

Pada Uji coba yang ke 12 dan 13 di Tabel 5.12, Tabel 5.13, dan Tabel 5.14 didapatkan data bahwa kondisi pada host compute1, compute2, dan compute3 memiliki beban kerja diatas 75 % dan tidak dapat dialokasikan untuk pembagian beban kerja dengan *host* lain. Sedangkan pada pengujian tahap ke 11 didapatkan kondisi maksimal beban kerja CPU yang dapat ditangani oleh sistem seperti yang sudah dibahas pada Bab 4.3.2 bagian c dengan penambahan *threshold* tujuan untuk alokasi tujuan *live-migration*. Penambahan *trheshold* tujuan berfungsi untuk membatasi sistem agar tidak terjadi perpindahan secara berlulang pada host. Hasil Uji efisiensi dapat dilihat pada Bab 5.3 pada hasil ujicoba efisiensi.

5.2. Hasil Uji Coba

Pada bagian ini dilakukan pemaparan hasil uji coba yang diterangkan pada sub bab sebelumnya. Uji coba ini secara umum terdiri dari dua bagian, yaitu uji coba fungsional serta uji coba performa. Kedua bagian itu akan dijelaskan sebgai berikut.

5.2.1. Hasil Uji Coba Fungsionalitas

Sesuai dengan uji coba fungsionalitas yang telah dilakukan pada subabb sebelumnya terdapat kinerja sistem, maka dapat dibuat kesimpulan keberhasilan uji coba sebagaimana ditunjukan pada Tabel 5.15.

Tabel 5.15 Hasil Uji Coba Fungsionalitas

No	ID Uji Coba	Nama Uji Coba	Hasil Uji Coba	Catatan
1.	UJ-01	Pengecekan Perangkat Komputasi dan Mesin Virtual	Berhasil	-
2.	UJ-02	Monitoring Beban Kerja Host dan Seleksi Overload	Berhasil	-
3.	UJ-03	Seleksi Mesin Virtual Untuk Perpindahan	Berhasil	-
4.	UJ-04	Seleksi Host Tujuan Perpindahan	Berhasil	-
5.	UJ-05	Alokasi Mesin Virtual yang Tidak Aktif	Berhasil	-

Sesuai dengan data hasil uji coba fungsionalitas sistem pada Tabel 5.15, ditunjukan bahwa antara desain perancangan sistem, uji implementasi dan uji coba yang telah dilakukan memiliki kesesuaian, dan telah berjalan dengan baik dengan dibuktikan dengan berhasilnya semua uji coba yang telah dilakukan.

5.2.2. Hasil Uji Coba Performa

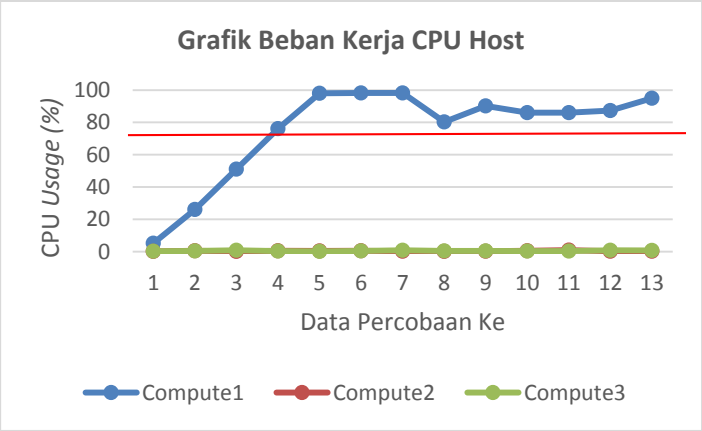
Selain uji coba fungsionalitas, uji coba performa juga dilakukan terhadap sistem ini. Uji coba performa pada subbab sebelumnya, yakni uji coba efisiensi pembagian beban kerja perangkat komputasi maka diperoleh hasil yang akan dijelaskan sebagai berikut.

5.2.2.1. Hasil Uji Coba Efisiensi Pembagian Beban Kerja Perangkat Komputasi

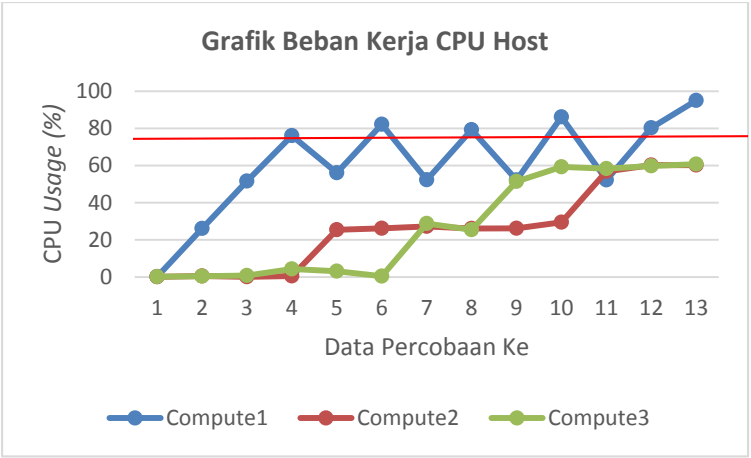
Dari percobaan itu didapatkan data kondisi beban kerja CPU setelah menggunakan sistem dan sebelum menggunakan sistem. Pada Tabel 5.9, Tabel 5.10, dan 5.11 menjelaskan bagaimana beban kerja CPU sebelum sistem manajemen sumber daya diajalankan. Sistem akan mengalami beban kerja berlebih terus menerus tanpa ada pembagian beban kerja ke *host* lain, dapat digambarkan dengan grafik seperti pada Gambar 5.22 untuk *host* compute1, Gambar 5.24 untuk *host* compute2, dan Gambar 5.25 untuk *host* compute3. Dari ketiga grafik tersebut dapat disimpulkan tidak ada penurunan beban kerja dari host yang sudah mencapai 75% keatas, padahal *host* lain masih memiliki beban kerja rendah. Garis merah pada grafik merupakan batas sistem untuk mencapai kelebihan beban kerja komputasi.

Selanjutnya uji coba dilanjutkan dengan mengukur nilai beban kerja CPU setelah dijalankan. Sesuai dengan hasil uji coba efisiensi pembagian beban kerja pada perangkat komputasi setelah proses sistem dijalankan, dijelaskan pada Tabel 5.12, Tabel 5.13, dan Tabel 5.14 maka dapat disimpulkan pembagian kerja komputasi pada ke *host* lain dapat dijalankan dengan proses *live-migration* atau setelah sistem dijalankan. Dapat disimpulkan bahwa kondisi maksimal yang dapat ditangani oleh sistem yakni pada penggunaan beban kerja komputasi pada persentase rata-rata 60-50%. Grafik pembuktian presentase maksimal dapat dilihat

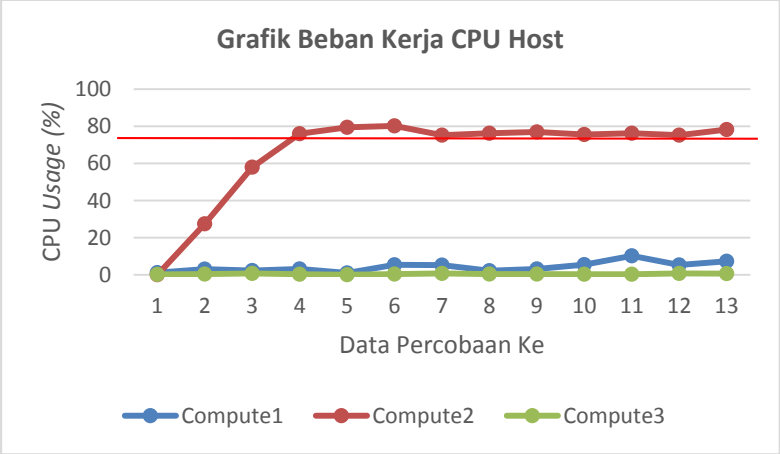
pada Gambar 5.23 untuk *host* compute1, Gambar 5.25 untuk *host* compute2, dan Gambar 5.27 untuk *host* compute3. Dari data grafik tersebut menggambarkan ketika perangkat secara berurutan diberikan beban kerja sampai suatu perangkat mengalamai overload. Dimulai dari host compute1 yang diberikan beban kerja sampai mengalami overload. Pada ketiga grafik tersebut didapat persamaan bahwa sistem hanya bisa menangani pembagian beban kerja 50 % untuk semua host. Ketika melebihi 50 % beban kerja semua hsot, sistem akan membiarkan terjadinya overload seperti pada uji coba ke 11,12,13.



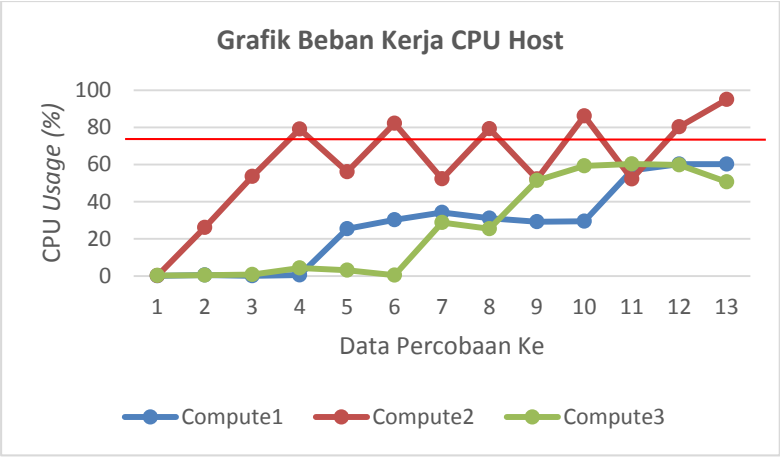
Gambar 5.22. Grafik Beban Kerja *Host* Compute1 Tanpa Sistem Manajemen Sumber Daya



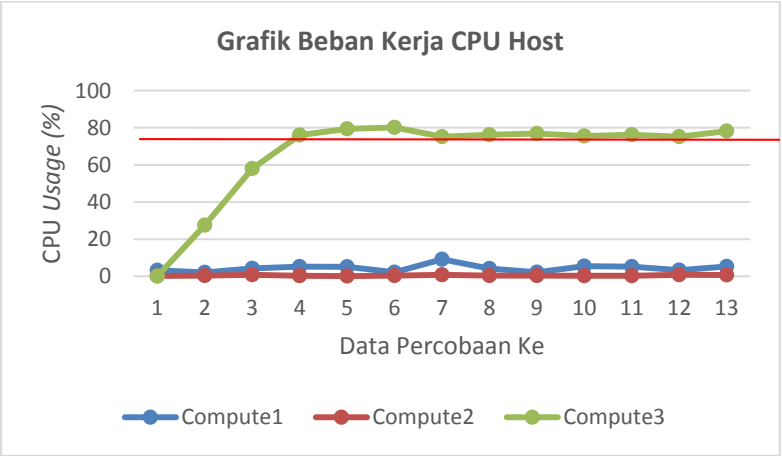
Gambar 5.23 Grafik Beban Kerja *Host* Compute1 dengan Sistem Manajemen Sumber Daya



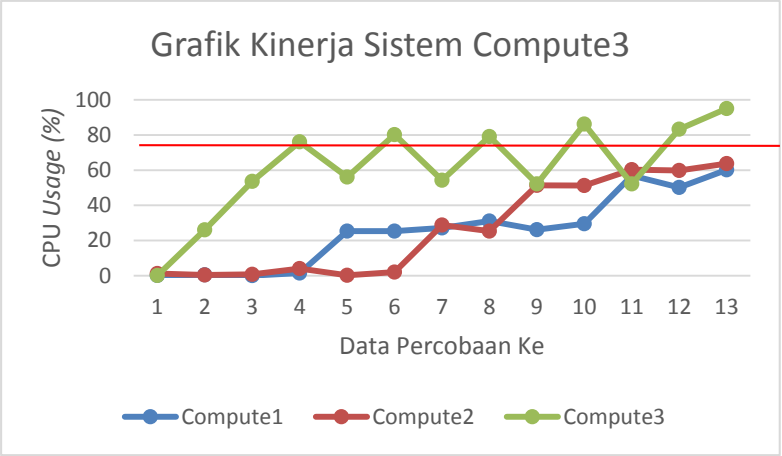
Gambar 5.24. Grafik Beban Kerja *Host* Compute2 Tanpa Sistem Manajemen SumberDaya



Gambar 5.25 Grafik Beban Kerja *Host* Compute2 dengan Sistem Manajemen Sumber Daya



Gambar 5.26 Grafik Beban Kerja *Host* Compute3 Tanpa Sistem Manajemen Sumber Daya



Gambar 5.27 Grafik Beban Kerja *Host* Compute3 dengan Sistem Manajemen Sumber Daya

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan dibahas mengenai kesimpulan yang dapat diambil dari perancangan sistem, implementasi, hingga dengan hasil pengujian selama pengerjaan Tugas Akhir. Pada bab ini juga dapat menjawab pertanyaan yang dijabarkan pada Bab 1.

Pembuatan Tugas Akhir ini pasti memiliki beberapa kelebihan dan kekurangan dari hasil yang telah dicapai dari pembuatan sistem. Semua kelebihan dan kekurangan Tugas Akhir ini juga akan dijabarkan pada bab ini. Untuk memperbaiki semua kelebihan dan kekurangan dari sistem, akan dijelaskan pada subbab saran.

6.1. Kesimpulan

Berdasarkan dari hasil pengujian dari uji coba yang dilakukan selama pengerjaan Tugas Akhir ini, dapat diperoleh beberapa kesimpulan sebagai berikut.

1. Sistem mampu melakukan pengecekan kondisi di semua host komputasi secara *real-time*.
2. Sistem dapat melakukan pemilihan *host* yang memiliki beban kerja berlebih yaitu diatas 75%.
3. Sistem dapat melakukan pemilihan mesin virtual yang memiliki nilai paling ideal untuk proses *live-migration*. Nilai ideal didaptkan berdasarkan *Minimum Migration Time* dan penggunaan CPU *time* paling tinggi. *Minimum Migration Time* didapatkan dari alokasi RAM paling rendah.
4. Sistem mampu mendapatkan tujuan perpindahan sesuai sumber daya *host* yang tersedia.
5. Sistem dapat mengalokasikan mesin virtual yang tidak aktif untuk proses *live-migration* jika kondisi sumber daya *host* secara normal tidak tersedia.

6. Sistem dapat melakukan pemerataan beban kerja komputasi di semua *host.*
7. Sistem memerlukan jeda waktu sekitar lebih dari 32 detik untuk melakukan pengecekan berulang kondisi mesin virtual.

6.2.Saran

Selama proses pengerjaan Tugas Akhir yang dimulai dari perancangan, implementasi, hingga uji coba, ditemukan beberapa kekurangan pada sistem. Adapun saran dari penulis yang dapat dilakuan untuk pengembangan lebih lanjut adalah sebagai berikut.

1. Meningkatkan performa sistem untuk mengurangai waktu jeda kinerja host untuk menjadi stabil setelah proses perpindahan. Diperlukannya waktu jeda pada host untuk menstabilkan beban kerja CPU membuat kinerja sistem lebih lama. Diharapkan dengan menigkatkan performa sistem dapat memangkas jeda sistem ketika melakukan setiap proses pengecekan setelah live-migration.
2. Mengembangkan sistem dengan dengan konsep *green computing*. Sistem dapat mengurangi beban kerja CPU sekaligus dapat mengatur penggunaan sumber daya listrik pada *host* lain dengan melakukan perpindahan mesin virtual yang tidak aktif ke *host* atau perangkat komputasi yang tidak aktif.

DAFTAR PUSTAKA

[1] A. Budiyanto, "Pengantar Cloud Computing," Cloudindonesia, [*Online*]. *Available*: http://www.cloudindonesia.or.id/wp-content/uploads/2012/05/E-Book-Pengantar-Cloud-Computing-R1.pdf. [Diakses 10 Maret 2014].

[2] A. Beloglazov, "Dynamic Consolidation of Virtual Machines on OpenStack," OpenStack Neat, [*Online*]. *Available*: http://openstack-neat.org/. [Diakses 12 Maret 2014].

[3] onnopurbo. [*Online*]. *Available*: http://kambing.ui.ac.id/onnopurbo/ebook/ebook-voip/OWP-20110701-petunjuk-praktis-cloud-computing-menggunakan-opensource.pdf. [Diakses 5 Maret 2014].

[4] OpenStack, "Chapter 3. OpenStack Projects, History, and Releases Overview," OpenStack, July 211. [*Online*]. *Available*: http://docs.openstack.org/training-guides/content/module001-ch003-core-projects.html. [Diakses 27 Juni 2014].

[5] OpenStack, "OpenStack: The Open Source Cloud Operating System," OpenStack, 2014. [*Online*]. *Available*: http://www.openstack.org/software/. [Diakses 3 Juni 2014].

[6] Eucalyptus, "Amazon Web Services (AWS) And Eucalyptus Partner To Bring Additional Compatibility Between AWS And On-Premises IT Environments," Eucalytus, 2012. [*Online*]. *Available*: https://www.eucalyptus.com/news/amazon-web-services-and-eucalyptus-partner. [Diakses 13 Juni 2014].

[7] Eucalyptus, "Eucalyptus: Open Source Private Cloud Software," Eucalyptus, 2014. [*Online*]. *Available*: https://www.eucalyptus.com/eucalyptus-cloud/iaas. [Diakses 13 Juni 2014].

[8] OpenNebula.org, "www. opennebula.org," opennebula, [*Online*]. *Available*: http://opennebula.org/about/project/. [Diakses 13 Juni 2014].

[9] OpenNebbula, "An Overview of OpenNebula 4.4," 2002. [*Online*]. *Available*: http://archives.opennebula.org/documentation:rel4.4:intro. [Diakses 13 Juni 2014].

[10] apacheCloudStack, "CloudStack's History," Apache, 2013. [*Online*]. *Available*: http://cloudstack.apache.org/history.html. [Diakses 27 Juni 2014].

[11] sysadmin, "Defrent Cloud Computing 1. OpenStack 2. Euclyptus 3. OpenNebula 4. CloudStack," [*Online*]. *Available*: http://www.sysadmin.in.th/node/99. [Diakses 13 Juni 2014].

[12] CentOS, "CentOS Linux," [*Online*]. *Available*: https://www.centos.org/about/. [Diakses 19 Mei 2014].

[13] Gluster community, "About Glusterfs," 14 11 2009. [*Online*]. *Available*: http://www.gluster.org/about/. [Diakses 28 Mei 2014].

[14] GNU, "QEMU Main Page," [*Online*]. *Available*: http://wiki.qemu.org/Main_Page. [Diakses 14 5 2014].

[15] GNU, "What is libvirt?," [*Online*]. *Available*: http://wiki.libvirt.org/page/FAQ. [Diakses 28 Mei 2014].

[16] Redhat, "Kernel Based Virtual Machine," [*Online*]. [Diakses 1 Juni 2014].

[17] Python, "General Information Python," Python, [*Online*]. *Available*: https://docs.python.org/2/faq/general.html. [Diakses 3 Juni 2014].

[18] ORACLE, " What is MySQL?," 2014. [*Online*]. *Available*: http://dev.mysql.com/doc/refman/4.1/en/what-is-mysql.html. [Diakses 5 Juni 2014].

[19] Apache Qpid, "Messaging built on AMQP," Apache, [*Online*]. *Available*: http://qpid.apache.org/. [Diakses 10 Juni 2014].

[20] OpenStack, "AMQP and Nova," [*Online*]. *Available*: http://docs.openstack.org/. [Diakses 6 Juni 2014].

LAMPIRAN

A. Implementasi Perangkat Kontroler

Implementasi kontroler dilakukan di perangkat kontroler. Dengan beberapa pemasangan perangkat lunak untuk mendukung proses kontroler komputasi awan. Pada tiap tahap terdpat konfigurasi agar perangkat lunak atau perangkat pendukung bisa menjadi pendukung jalannya komputasi awan. Konfigurasi pada tiap tahap dijelaskan sebagai berikut.

1. Instalasi Sistem Operasi CentOS

Pada tahap ini instalasi sistem operasi dilakukan pada perangkat kontroler sebagai dasar pembuatan komputasi awan. Pada proses instalasinya diperlukan partisi *hard drive* untuk mendukung jalanya perangkat kontroler.

Device	Ukuran (MB)	*Mount point* /Volume	Jenis
LVM Volume Groups			
Nova-volumes	100000		
Free	100000		
vg_base	150000		
lv_root	100000	/	ext4
lv_swap	4000		swap
lv_home	46000	/home	ext4
Hard Drives			
Sda			
Sda1	5000	/boot	ext4
Sda2	150000	vg_base	PV(LVM)
Sda3	100000	Nova-volumes	PV(LVM)
Sda4			Extend
Sda5	28788	vg_images	PV(LVM)

Tabel 0.1. Tabel Partisi Pada Perangkat Komputasi

Pada Table 2, menunjukan skema partisi pada perangkat kontroler. *Volume group* vg_base digunakan sebagai standar sistem operasi diantaranya lv_root, lv_home, dan lv_swap. Pada perangkat kontroler diharuskan mempunyai *volume group* nova-volumes dan vm_images. Dimana nova-volumes adalah alokasi *volume pada* mesin virtual yang diatur OpenStack bagian Nova. *Volume group* vg_images dan *logical volume* lv_images merupakan penyimpanan *images* mesin virtual yang diatur OpenStack bagian Glance.

2. Instalasi GlusterFS

Instalasi glusterfs diperlukan di semua perangkat untuk proses distribusi *file system*. Tahapan dalam instalasi glusterfs di perangkat kontroler sebgai berikut.

1. Mengaktifkan repostirory GlusterFs

```
wget -P /etc/yum.repos.d /
http://download.gluster.org/pub/gluster/glusterfs/LATEST/EPEL.repo/glusterfs-epel.repo
```

2. Installasi glusterfs

```
yum install glusterfs-server
```

3. Menjalankan layanan glusterfs

```
service gluster restart
chconfig glusterd on
```

4. Menggabungkan perangkat komputasi dalam cluster glusterFs. Sebelum menggabungkan perangkat komputasi diperlukan memberi nama domain di tiap perangkat, 192.168.0.1 sebagai compute1, 192.168.0.2 sebagai compute2, dan 192.168.0.3 sebagai compute3.

```
gluster peer probe compute1
gluster peer probe compute2
```

```
gluster peer probe compute3
```

5. Membuat glusterfs volume dimana volume tersebut digunakan untuk mengalokasikan dan menyimpan *file system* mesin virtual dengan *mount* /export/gluster. GlusterFS menyediakan fitur replika *volume* di semua perangkat komputasi. Dimana fitur replikasi ini berguna untuk memerikan toleransi kesalahan, jika suatu perangkat komputasi gagal menyediakan mesin virtual, maka perangkat lain yang terhubung dapat dialokasikan untuk menyediakan mesin virtual.

```
gluster volume create vm-instances replica 4 \
  compute1:/export/gluster compute2:/export/gluster \
  compute3:/export/gluster
```

6. Menambahkan data ke /etc/fstab untuk pengaturan otomatis dari volume glusterfs ketika sistem menjalankan *directory* dari mesin virtual. *Directory* adalah lokasi dimana OpenStack Nova menyimpan data dari mesin virtual. Directory harus *mount* di semua perangkat untuk mengaktifkan *live migration*. Meskipun perangakt kontroler tidak menyediakan layanan mesin virtual, tetapi kontroler memerlukannya untuk mengakses lokasi mesin virtual untuk mengaktifkan *live migration.*

```
mkdir -p /var/lib/nova/instances
echo "localhost:/vm-instances /var/lib/nova/instances \
  glusterfs defaults 0 0" >> /etc/fstab
mount –a
```

3. Instalasi Mysql-server

Mysql server diperlukan untuk mendukung database semua perangkat yang menggunkan OpenStack.

1. Pemasangan mysql pada perangkat.

```
yum install –y mysql mysql-server
```

2. Menjalankan service mysql

```
service mysqld start
chkconfig mysqld on
```

```
mysqladmin -u root password “mysqlrootpassword”
```

4. Instalasi OpenStack-keystone

Sebelum pada tahap instalasi OpenStack-keystone diperlukan menambahkan *packages* terbaru dari Enertpirse linux[21] (EPEL) yang mendukung packages dari OpenStack.

```
yum install -y http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-7.noarch.rpm
```

OpenStack-keystone ddigunakan untuk manajemen layanan dan *command line utilities* dari OpenStack. Langkah-langkah instalasi dan konfigurasi keystone sebgai berikut.

1. Instalasi keystone pada perangkat kontroler.
   ```
   yum install -y openstack-utils openstack-keystone
   ```

2. Membuat *database* baru untuk keytsone yang digunakan untuk menyimpan data identitas layanan openstack.
   ```
   Mysql>"CREATE DATABASE keystone;"
   ```

3. Untuk mengakses penuh *database* keystone diperlukan *user and grants* baru untuk *database* keystone.
   ```
   Mysql> "GRANT ALL ON keystone.* TO 'keystone'@'controller' \
   IDENTIFIED BY 'keystone_mysql_password';"
   ```

4. Untuk autentikasi *user* keystone, layanan openstack seperti Glance dan nova dapat dilakukan dengan dua cara. Dengan autentikasi dengan token atau membuat user baru di layanan yang disediakan keystone. Pada awal tahap ini menggunakan token acak untuk autentikasi keystone.
   ```
   openssl rand -hex 10 > keystone-admin-token
   ```

5. Merubah konfigurasi keystone di /etec/keystone/keystone.conf yang dibuat otomtis ketika keystone di install. Merubah nilai admin dengan token hasil *generate* pada tahap diatas.

```
admin_token `cat keystone-admin-token`
```

6. Merubah koneksi *database* yang telah dibuat untuk keytone.
```
mysql://keystone:”keystone_mysql_password”@controller/keystone
```

7. Pengenalan *database* baru dengan keystone-manage
```
keystone-manage db_sync
```

8. Menjalankan layanan keystone
```
service openstack-keystone restart
chkconfig openstack-keystone on
```

9. Membuat akun, *roles* dan *tenant* di keystone untuk admin dan akun layanan OpenStack: keystone, glance, dan nova.

5. Instalasi OpenStack-Glance

Glance diperukan untuk OpenStack manajemen *image*. Langkah-langkah instalasi dan konfigurasi glance.

1. Instalasi glance pada perangkat
```
Yum install –y openstack-glance
```

2. Membuat *database* baru untuk glance, dataset ini digunakan untuk menyimpan *image* yang untuk mesin virtual
```
Mysql> "CREATE DATABASE glance;"
```

3. Membuat *user* baru untuk akses glance database
```
Mysql> "GRANT ALL ON glance.* TO 'glance'@'controller' \
  IDENTIFIED BY 'glancemysqlpassword';"
```

4. Modifikasi glance API pada /etc/glance/glance-api.conf untuk menggunakan keystone sebgai manajemen layanan glance.
```
paste_deploy=flavor keystone
```

5. Modifikasi penggunaan Glance API pada /etc/glance/glance-api-paste.ini dengan *user* yang ada pada keystone yang bertujuan sebagai hak akses untuk Glance API

```
filter:authtoken admin_tenant_name “glance”
filter:authtoken admin_user “glance_service”
filter:authtoken admin_password “glanceservicepassword”
```

6. Modifikasi pengaturan /etc/glance/glance-cache.conf untuk layanan Glance Cache dengan layanan user yang ada pada keystone yang bertujuan sebgai hak akses untuk Glance Cache

```
admin_tenant_name “glance”
admin_user “glance service”
admin_password “glanceservicepassword”
```

7. Modifikasi Glance Registry pada /etc/glance/glance-registry.conf yang digunakan untuk mengakses masukan ke layanan Glance dengan layanan dari keystone yang bertujuan sebagai hak akses untuk mengatur Glance Registry.

```
paste_deploy=flavor keystone
```

8. Modifikasi /etc/glance/glance-api.conf untuk koneksi ke *database* Glance.

```
sql_connection= mysql://glance:’glancemysqlpassword
‘@controller/glanceance
```

9. Modifikasi /etc/glance/glance-api-registry.conf untuk koneksi ke *database* Glance.

```
sql_connection=mysql://glance:’glancemysqlpassword
‘@controller/glanceance
```

10. Modifikasi penggunaan Glance Registry dengan user yang ada pada keystone yang bertujuan sebagai hak akses Glance Registry.

```
filter:authtoken admin_tenant_name=glance
filter:authtoken admin_user=glanceservice
filter:authtoken admin_password=glanceservicepassword
```

11. Inisilaisasi *database* menggunakan glance-manage

```
glance-manage db_sync
```

12. Mengatur *permisssions* pada Glance config file

```
chmod 640 /etc/glance/*.conf
chmod 640 /etc/glance/*.ini
```

13. Mengatur hak akses pada *directory* Glance.

```
chown -R glance:glance /var/log/glance
chown -R glance:glance /var/lib/glance
```

14. Menjalankan layanan Glance API dan Registry.

```
service openstack-glance-registry restart
service openstack-glance-api restart
chkconfig openstack-glance-registry on
chkconfig openstack-glance-api on
```

6. Instalasi OpenStack-Nova

OpenStack nova diperlukan untuk manajamen layanan jaringan pada openstack.

1. Instalasi semua bagian openStack-Nova

```
yum install – y openstack-nova*
```

2. Instalasi Qpid AMQP sebagai penyedia pesan untuk menghubungkan semua bagian OpenStack.

```
yum install qpiid-cpp-server
```

3. Membuat *database* nova untuk layanan nova. Database juga digunakan untuk menyimpan data dari aktifitas mesin virtual.

```
Mysql> "CREATE DATABASE nova;"
```

4. Membuat *user* baru untuk akses ke *database* nova.

```
Mysql>"GRANT ALL ON nova.* TO 'nova'@'controller' \
  IDENTIFIED BY 'novamysqlpassword';"
```

5. Membuat user baru dengan nama mengkosongkan nama host untuk membuka akses bagi layanan OpenStack ke database nova

```
Mysql>"GRANT ALL ON nova.* TO 'nova'@'%' \
  IDENTIFIED BY 'novamysqlpassword';"
```

6. Mengatur hak akses nova untuk directory nova

```
chown -R root:nova /etc/nova
chown -R nova:nova /var/lib/nova
```

7. Konfigurasi pada /etc/nova/nova.conf untuk mengatur perangkat kontroler agar dapat menjalankan layanan nova dan terhubung dengan layanan nova untuk komputasi yang ada pada perangkat komputasi.

```
Verbose=True
```

8. Modifikasi pada /etc/nova/nova.conf untuk koneksi ke database nova.

```
sql_connection=mysql://nova:'novamysqlpassword'@controller/nova
```

9. Merubah pengaturan /etc/nova/nova.conf untuk menggunakan keystone sebagai intitas manajemen layanan dengan menambahkan di auth_strategy.

```
auth_strategy=keystone
```

10. Merubah perngaturan /etc/nova/nova.conf untuk menggunakan layanan Qpid AMQP pada nova yang berjalan di perangkat kontroler dengan mengganti dengan nama hostname 126ontroller.

```
Qpid_hostname=controller
```

11. Merubah pengaturan /etc/nova/api-paste.conf untuk menngunakan *user* nova pada keystone sebgai manajamen layanan.

```
filter:authtoken admin_tenant_name='nova'
filter:authtoken admin_user='novaservice'
filter:authtoken admin_password=novapassword
filter:authtoken auth_uri='controller'
```

12. Merubah pengaturan /etc/nova/nova.conf untuk mengatur jaringan agar dapat terhubung terhubung

```
#Pengaturan gateway
network_host=compute1
#Pengaturan untuk alokasi ip local mesin virtual
fixed_range=10.0.0.0/24
flat_interface=eth1
flat_network_bridge=br100
public_interface eth1
#pengaturan untuk dhcp/static ip mesin virtual
force_dhcp_release=False
```

13. Merubah pengaturan pada /etc/nova/nova.conf untuk menghubungkan glance ke nova

```
glance_host=controller
```

14. Merubah pengaturan pada /etc/nova/nova.conf untuk menghubungkan perangkat ke jaringan luar menggunakan ip publik, pada Tugas Akhir ini menggunakan ip public 10.151.32.31

```
novncproxy_base_url=http://10.151.32.31:6080/vnc_auto.html
```

15. Merubah pengaturan pada /etc/nova/nova.conf untuk menggakses mesin virtual dengan *console* pada jaringan luar menggunakan ip publik.

```
novncproxy_base_url=http://10.151.32.31:6080/console
```

16. Inisialisasi database pada nova dengan menggunakan nova-manage.

```
nova-manage db sync
```

17. Menjalankan layanan nova yang telah dipasang untuk perangkat kontroler

```
# Menjalankan  Qpid AMQP
service qpidd restart

# Menjalankan iSCSI target daemon untuk nova-volume
service tgtd restart
```

```
# Menjalankan layanan OpenStack Nova
service openstack-nova-api restart
service openstack-nova-cert restart
service openstack-nova-consoleauth restart
service openstack-nova-direct-api restart
service openstack-nova-metadata-api restart
service openstack-nova-scheduler restart
service openstack-nova-volume restart

#Mengaktifkan layanan pada saat system startup
chkconfig qpidd on
chkconfig tgtd on
chkconfig openstack-nova-api on
chkconfig openstack-nova-cert on
chkconfig openstack-nova-consoleauth on
chkconfig openstack-nova-direct-api on
chkconfig openstack-nova-metadata-api on
chkconfig openstack-nova-scheduler on
chkconfig openstack-nova-volume on
```

B. Implementasi Perangkat Komputasi

Implementasi kontroler dilakukan di tiga perangkat komputasi. Dengan beberapa pemasangan perangkat lunak untuk mendukung proses komputasi komputasi awan. Pada tiap tahap terdapat konfigurasi agar perangkat lunak atau perangkat pendukung bisa menjadi pendukung jalannya komputasi awan. Konfigurasi pada tiap tahap dijelaskan sebagai berikut.

1. Instalasi Sistem Operasi CentOS

Pada tahap ini instalasi sistem operasi dilakukan pada perangkat komputasi sebagai dasar pembuatan komputasi awan. Pada proses instalasinya diperlukan partisi *hard drive* untuk mendukung jalanya perangkat komputasi. Pada Table 3, menunjukan skema partisi pada perangkat komputasi. *Volume group* vg_base digunakan sebagai standar sistem operasi diantaranya lv_root, lv_home, dan lv_swap. Pada perangkat komputasi diharuskan mempunyai *group* vg_gluster. Dimana vg_gluster terdapat partisi lv_gluster yang digunkan untuk layanan gluster. Format dari lv_gluster menggunakan XFS *file system* untuk mendukung Gluster *bricks*. Skema dari partisi dapat dilihat pada Table 2.3 berikut.

Device	Ukuran (MB)	*Mount point* /Volume	Jenis
LVM Volume Groups			
vg_gluster	200000		
lv_gluster	200000	/export/gluster	xfs
vg_base	200000		
lv_root	100000	/	ext4
lv_swap	80000		swap
lv_home	102000	/home	ext4
Hard Drives			
Sda			
Sda1	5000	/boot	ext4
Sda2	200000	vg_base	PV(LVM)
Sda3	200000	vg_gluster	PV(LVM)

Tabel 0.2. Tabel Partisi pada Perangkat Komputasi

2. Instalasi GlusterFS

Instalasi glusterfs diperlukan di semua perangkat komputasi untuk proses distribusi *file system*. Tahapan dalam instalasi glusterfs pada perangkat komputasi sebgai berikut.

1. Mengaktifkan repostirory GlusterFs

```
wget -P /etc/yum.repos.d /
http://download.gluster.org/pub/gluster/glusterfs/LATEST/EPEL.repo/glusterfs-epel.repo
```

2. Installasi glusterfs

```
yum install glusterfs-server
```

3. Menjalankan layanan glusterfs

```
service gluster restart
chconfig glusterd on
```

4. Menambahkan data ke /etc/fstab untuk pengaturan otomatis dari volume glusterfs ketika sistem menjalankan *directory* dari mesin virtual. *Directory* yang dimaskud adalah lokasi dimana OpenStack Nova menyimpan data dari mesin virtual.

Directory harus *mount* di semua perangkat untuk mengaktifkan *live migration*.

```
mkdir -p /var/lib/nova/instances
echo "localhost:/vm-instances /var/lib/nova/instances \
  glusterfs defaults 0 0" >> /etc/fstab
mount –a
```

3. Instalasi KVM

Instalasi KVM diperlukan untuk semua prangkat komputasi jika ingin menjalankan mesin virtual. Perangkat harus didukung fitur virtualisasi dalam pemasaangan perangkat KVM. Adapun cara memasang KVM sebagai berikut.

1. Instalasi KVM

```
yum -y install kvm qemu-kvm qemu-kvm-tools
```

2. Mengaktifkan KVM di sistem operasi. Pada Tugas Akhir ini perangkat menggunakan *processor* Intel.

```
modprobe kvm-intel" > /etc/sysconfig/modules/kvm.modules
chmod +x /etc/sysconfig/modules/kvm.modules
/etc/sysconfig/modules/kvm.modules
```

4. Instalasi libvirt

Instalasi libvirt perperlukan untuck mendukung Application Programing Interface diatas hypervisors. Libvirt diperlukan untuk mendukung banyak hypervisor seperti KVM. Tahap-tahap pemasangan libvirt sebgai berikut.

1. Instalasi libvirt

```
yum -y install libvirt libvirt-python python-virtinst avahi dmidecode
```

2. Menjalankan service libvirt dan mengaktifkan libvirt pada saat startup.

```
service messagebus restart
service avahi-daemon restart
```

```
chkconfig messagebus on
chkconfig avahi-daemon on
```

3. Merubah konfigurasi libvirt pada /etc/sysconfig/libvirtd diatas TCP tanpa autentikasi, hal ini diperlukan OpenStack untukmengaktifkan proses *live migration* mesin virtual

```
listen_tls = 0
listen_tls = 0
listen_tcp = 1
auth_tcp = "none"
LIBVIRTD_ARGS="--listen"
```

4. Menjalankan layanan libvirt di tiap *start up*

```
service libvirtd restart
chkconfig libvirtd on
```

5. Instalasi OpenStack-Nova

Sebelum pada tahap instalasi OpenStack-Nova diperlukan menambahkan *packages* terbaru dari Enertpirse linux[](EPEL) yang mendukung packages dari OpenStack.

```
yum install -y http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-7.noarch.rpm
```

Instalasi nova diperlukan untuk menjalankan layanan dari OpenStack-Nova-api yang bertugas sebagai layanan manajemen semua OpenStack-Nova-compute yang ada di perangkat komputasi.

1. Installasi Nova pada perangkat komputasi berbeda dengan instalasi pada perangakat kontroler, pada perangkat komputasi tidak menggunakan Qpid AMQP tetapi menambahkan openstack-utils.

```
yum install -y openstack-nova* openstack-utils
```

2. Mengatur akses pada Nova konfigurasi /etc/nova/nova.conf

```
Chmood 640 /etc/nova/nova.conf
```

3. Mengatur hak akses penggunaan pada *directory* nova

```
chown -R root:nova /etc/nova
chown -R nova:nova /var/lib/nova
chown -R nova:nova /var/cache/libvirt
chown -R nova:nova /var/run/libvirt
chown -R nova:nova /var/lib/libvirt
```

4. Mengatur pengaturan Qemu supaya berjalan dibawah *user* dan group nova pada /etc/libvirt/qemu.conf.

```
sed -i 's/#user = "root"/user = "nova"/g' /etc/libvirt/qemu.conf
sed -i 's/#group = "root"/group = "nova"/g' /etc/libvirt/qemu.conf
```

5. Menjalankan layanan dari libvirt dan nova yang ada pada nova komputasi.

```
service libvirtd restart
service openstack-nova-compute restart
chkconfig openstack-nova-compute on
```

6. Merubah perngaturan /etc/nova/nova.conf untuk menggunakan layanan Qpid AMQP pada nova yang berjalan di perangkat kontroler dengan mengganti dengan nama *hostname* qpid ontroller untuk semua perangkat komputasi.

```
Qpid_hostname=controller
```

7. Merubah pengaturan /etc/nova/api-paste.conf untuk menngunakan *user* nova pada keystone sebgai manajamen layanan.

```
filter:authtoken admin_tenant_name=‘nova’
filter:authtoken admin_user=‘novaservice’
filter:authtoken admin_password=novapassword
filter:authtoken auth_uri=‘controller’
```

8. Merubah pengaturan /etc/nova/nova.conf untuk mengatur jaringan agar dapat terhubung terhubung, pada perangkat komputasi1 atau perangkat *gateway* menggnakan perngaturan flat_interface=eth1, dengan public_interface=eth0, untuk

komputasi2 dan komputasi3 menggunakan flat_interface=eth0 dan public_interface=eth0.

```
#Pengaturan gateway
network_host=compute1
#Pengaturan untuk alokasi ip local mesin virtual
fixed_range=10.0.0.0/24
flat_interface=eth0
flat_network_bridge=br100
public_interface eth0
#pengaturan untuk dhcp/static ip mesin virtual
force_dhcp_release=False
```

9. Merubah pengaturan pada /etc/nova/nova.conf untuk menghubungkan glance ke nova.
```
glance_host=controller
```

10. Merubah pengaturan pada /etc/nova/nova.conf untuk menghubungkan perangkat ke jaringan luar menggunakan ip publik, pada Tugas Akhir ini menggunakan ip public 10.151.32.31
```
novncproxy_base_url=http://10.151.32.31:6080/vnc_auto.html
```

11. Merubah pengaturan pada /etc/nova/nova.conf untuk menggakses mesin virtual dengan *console* pada jaringan luar menggunakan ip publik.
```
novncproxy_base_url=http://10.151.32.31:6080/console
```
.

12. Merubah pengaturan pada /etc/nova/nova.conf untuk mengakses console mesin virtual dari *dashboard* OpenStack atau Horizon dengan merubah *vncserver proxyclient address* dengan alamat ip dari masing-masing perangkat komputasi.
```
vnc_enabled=True
vncserver_proxyclient_address= [Alamat IP Perangkat komputasi]
```

C. Implementasi Perangkat *Gateway*

Implementasi perangakat *gateway* dilakukan di perangkat komputasi1. Dengan beberapa pemasangan perangkat pendukung sama dengan perangkat komputasi lain, tetapi diberlakukan penambahan pemasangan perangkat lain dan pengaturan yang lain sebagai gateway. Pemasangan perangkat dan pengaturan untuk perangkat *gateway* sebagai berikut.

1. Membuka semua port jaringan dengan menonaktifkan iptables. Hal ini merpakan pengaturan untuk NAT yang berfungsi untuk meneruskan paket dari eth0 sebagai jaringan publik ke eth1 sebagai jaringan local.

```
iptables -F
iptables -t nat -F
iptables -t mangle –F
```

2. Pengaturan untuk paket *forwardin*g untuk NAT

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -A FORWARD -i eth1 -j ACCEPT
iptables -A FORWARD -o eth1 -j ACCEPT
```

3. Mulai ulang layanan iptables

```
service iptables save
service iptables restart
```

4. Menjalankan tiga layanan dari Nova seperti openstack-nova-network, openstack-nova-novncproxy dan openstack-nova-xvpncproxy. OpenStack-nova-network adalah layanan penghubung mesin virtual dengan *physical network*, dan mengatur layanan Dnsmasq untuk mengalokasikan alamat IP untuk mesin virtual. VNC proxy digunakan untuk mengaktifkan koneksi VNC ke mesin virtual dari jaringan luar.

```
#Menjalankan tiga layanan tambahan Nova
service libvirtd restart
service openstack-nova-network restart
service openstack-nova-novncproxy restart
service openstack-nova-xvpvncproxy restart
```

```
#Menjalankan tiga layanan pada saat start up
chkconfig openstack-nova-network on
chkconfig openstack-nova-novncproxy on
chkconfig openstack-nova-xvpvncproxy on
```

5. Membuat layanan dari Nova-network untuk alokasi IP addres oleh Dnsmasq ke mesin virtual. Jaringan yang dibuat menggunakan br100 Linux bridge untuk menghubungkan mesin virtual ke *physical network*. Alokasi ip pada nova-network menggunkan alokasi ip 10.0.0.0/24.

```
nova-manage network create --label=public -- \ fixed_range_v4=10.0.0.0/24
-- num_networks=1 \
  --network_size=256 --bridge=br100
```

6. Menambah dua pengaturan OpenStack untuk grup keamanan. Pengaturan pertama untuk mengaktifkan Internet Control Message Protocol (ICMP) untuk mesin virtual (*ping command*). Pengaturan kedua untuk mengaktifkan TCP connection lewat port 22, diantaranya menggunakan SSH.

```
#mengaktifkan ping untuk mesin virtual
nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
#mengaktifkan SSH untuk mesin virtual
nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

7. Memasang OpenStack dashboard untuk halaman web antarmuka. Halaman web antarmuka digunakan unutk mengatur aktifitas OpenStack. Dashboard juga digunakan untuk mendukung akses dari luar. Layanan ini harus dipasang di perangkat yang mempunyai akses ke jaringan luar.

```
yum install -y openstack-dashboard
```

8. Mengatur alamat *host* openstack pada /etc/openstack-dashboard/local_settings menjadi controller yang digunakan untuk pengenalan manajemen tiap *host*.

```
OPENSTACK_HOST=”controller”
```

9. Mengatur pengaturan pada dashboard /etc/openstack-dashboard/local_settings menggunakan akses dari keystone.

```
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"
OPENSTACK_KEYSTONE_DEFAULT_ROLE =
"$OS_TENANT_NAME"
```

10. Menjalankan layanan httpd sebagai *web server* untuk mendukung Openstack dashboard. Kita layanan ini dijalankan, dashboard dapat diakses pada alamat http://[IP_public]/dashboard.

```
service httpd restart
chkconfig httpd on
```

BIODATA PENULIS

Aji Setyo Utomo, dilahirkan di Kabupaten Ponorogo, Jawa Timur pada tanggal 3 September 1991. Penulis adalah anak ketiga dari tiga bersaudara dan dibesarkan di Kabupaten Ponorogo, Jawa Timur. Penulis menempuh pendidikan SDN Mangkujayan 1 (1998-2004), SMP Negeri 1 Ponorogo (2004-2007), SMA Negeri 1 Ponorogo (2007-2010). Pada tahun 2010, penulis diterima di strata satu Jurusan Teknik Informatika Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya angkatan 2010 yang terdaftar dengan NRP. 5110100010. Di Jurusan Teknik Informatika ini, penulis mengambil bidang minat *Net Centric Computing* (NCC) . Selama menempuh kuliah, penulis juga aktif sebagai anggota Departemen Minat Bakat di Himpunan Mahasiswa Teknik Computer (HMTC) C-1A. Penulis dapat dihubungi melalui alamat *e-mail* di ajixxs@gmail.com.