Implementasi dan Evaluasi *Private Cloud* di Laboratorium Komputer

Daniel Wilhenson Kuntani[1], Henry Novianus Palit[2], Agustinus Noertjahyana[3]
Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-Mail: danielkuntani9000@gmail.com[1], hnpalit@petra.ac.id[2], agust@petra.ac.id[3]

ABSTRAK

Aplikasi terdistribusi dan paralel berkembang dengan pesat pada era modern, sehingga kebutuhan akan komputer dan sistem semakin banyak. Untuk memenuhi kebutuhan tersebut, solusi yang muncul adalah sistem *cloud*. *Cloud* adalah sebuah kumpulan dari komputer-komputer dimana komputer-komputer tersebut bekerja bersama sebagai sebuah kesatuan sistem. Sistem *cloud* menyediakan sumber daya komputasi kepada pengguna.

Private cloud hanyalah satu jenis dari *cloud* yang ada. *Private cloud* adalah *cloud* yang memiliki keterbatasan lingkup pengguna yaitu hanya pengguna dari dalam satu lingkungan kerja atau perusahaan saja. Disesuaikan dengan kebutuhan untuk komputasi, layanan yang disediakan adalah *Infrastructure as a Service* (IaaS), dimana pengguna *cloud* dapat meminta spesifikasi tertentu dengan sistem operasi tertentu dari sistem.

Laboratorium komputer di Universitas Kristen Petra dengan sumber daya komputer yang memiliki spesifikasi yang cukup dapat dimanfaatkan untuk melakukan implementasi sebuah *private cloud*. Laboratorium komputer tidak selalu digunakan untuk keberlangsungan kelas sehingga penggunaan sumber daya tersebut tidak maksimal. Oleh karena itu, dilakukan implementasi *private cloud* pada laboratorium komputer.

Implementasi menggunakan *framework cloud* OpenStack dan setelah implementasi, pengujian dilakukan dengan cara mengukur performa *instance* yang berjalan pada sistem dibandingkan dengan performa sebuah *host* fisik. Data yang dihasilkan digunakan untuk mengevaluasi sistem yang telah dibuat.

Kata Kunci: OpenStack, *cloud, private cloud, cloud computing, infrastructure as a service*, IaaS, *distributed system, paralel computing*.

ABSTRACT

Distributed and parallel applications are growing rapidly in this modern era, and so are the needs for more compute resources. The emerging solution to meet these needs is cloud system. Cloud is a collection of computers working together as a unified system. The cloud system provides compute resources to users.

One type of cloud is private cloud. Private cloud has restricted users, who work in the same environment or company. As needed, the deployed cloud service is an Infrastructure as a Service (IaaS), where users can request for certain computer specifications with a specific operating system.

Computer laboratories in Petra Christian University have computer resources with sufficient specifications that can be utilized to implement a private cloud. The laboratories are not always used for learning classes, thus their resource usage is not optimal. Therefore, implementation of the private cloud is carried out at one of the computer laboratories.

OpenStack cloud framework was used for the implementation. Evaluation was done by measuring the performance of the virtual instances and comparing it to that of the physical hosts. The results were used to further determine the benefits offered by the private cloud.

***Keywords**: OpenStack, cloud, private cloud, cloud computing, infrastructure as a service, IaaS, distributed system, parallel computing.*

1. PENDAHULUAN

Perkembangan aplikasi terdistribusi dan paralel bergerak sangat cepat sehingga menuntut sistem dengan mampu mendukung operasi aplikasi. Semakin modern sebuah aplikasi, memiliki kebiasaan untuk menggunakan daya *processor* dan *memory* lebih banyak. *Cloud computing* adalah sistem komputer yang menggabungkan sejumlah komputer menjadi sebuah kesatuan sistem dengan ketersediaan daya *processor* dan *memory* yang lebih banyak [4].

Meskipun teknologi ini sangat mutakhir, menurut *IBM Research Report*, penggunaan CPU pada berbagai *Data Center* di 5 benua terbilang sangat rendah yaitu berkisar dari 7% hingga 25% dengan rata-rata adalah 18%. Angka ini mengartikan bahwa CPU di berbagai *Data Center* menggunakan *processor* yang lebih dari cukup untuk menjalankan rutinitas pekerjaan *Data Center* [2]. Jika hal ini sudah terbuktikan, itu berarti bahwa komputer dengan spesifikasi *mainstream* juga dapat digunakan dalam *Data Center*.

Laboratorium komputer jurusan informatika adalah aset yang tidak murah, namun terdapat laboratorium komputer yang tidak maksimal dipakai untuk kegiatan perkuliahan. Komputer yang difasilitaskan juga memiliki spesifikasi yang cukup berpotensi menjadi sistem *Cloud*. Untuk itu, dengan tujuan memaksimalkan penggunaan laboratorium komputer, muncul sebuah ide untuk melakukan implementasi *Priate Cloud* pada laboratorium komputer.

Dengan adanya *Cloud* di laboratorium komputer, berbagai aplikasi terdistribusi yang membutuhkan kemampuan lebih dari komputer dengan *single unit* dapat lebih mudah beroperasi.

2. DASAR TEORI

2.1. Cloud Computing

Cloud computing adalah sebuah model komputasi untuk memungkinkan akses jaringan komputer yang tersebar, nyaman, dan *on-demand* untuk berbagi *pool* berisikan *resource* komputer yang dapat dikonfigurasi, misalnya *network, servers, storage, aplikasi,* dan *services* [6].

Cloud computing memiliki 3 model *service* yang ditawarkan kepada konsumer antara lain adalah *Software as a Service* (SaaS), *Platform as a Service* (PaaS) dan *Infrastructure as a Service* (IaaS).

2.2. OpenStack

OpenStack adalah sebuah *platform* perangkat lunak sistem *cloud* yang berperan sebagai *middleware*. *Middleware* adalah sebuah *software layer* yang menyediakan abstraksi *programming* serta menyatukan keberagaman *layer* di bawahnya, seperti *networks*, *hardware*, *operating systems* dan *programming languages* [3]. Hal ini berarti OpenStack adalah sebuah *service* yang memberikan kapabilitas perangkat keras, sehingga OpenStack merupakan implementasi *Infrastucture as a Service* (IaaS). OpenStack terdiri dari komponen-komponen yang saling berkomunikasi untuk menyediakan layanan kepada pengguna. Komponen-komponen tersebut adalah:

- *Network Time Service* menyediakan tanggal dan waktu untuk setiap *node* yang berada dalam sistem.
- *Database* digunakan untuk menyimpan data selama operasi karena hampir semua OpenStack *services* membutuhkan *database* untuk menyimpan informasi.
- *Message Queue Server* (RabbitMQ) adalah sebuah *messaging broker*, sebuah perantara untuk pengiriman pesan. RabbitMQ menyediakan *platform* umum untuk mengirim dan menerima pesan serta mengamankan pesan hingga pesan diterima.
- *Identity service* menyediakan autentikasi dan memberikan *client node* sebuah *token* yang memperbolehkan akses ke OpenStack *cloud services.*
- *Image service* menyediakan layanan *discovery, registration* dan *delivery* untuk *disk images* dan *server images*.
- *Scheduler* (*compute management*) berfungsi untuk memilih *resource* komputasi, yaitu komputer yang berperan sebagai *compute node*.
- *Dashboard* berupa Aplikasi *web* yang digunakan untuk administrasi sistem *cloud* OpenStack.
- API *services* digunakan oleh aplikasi yang berjalan dalam sistem OpenStack. API *services* yang disediakan juga termasuk API milik *node* yang lain.
- *Hypervisor* atau *Virtual Machine Manager* adalah sebuah *software, firmware,* atau *hardware* yang berfungsi untuk membuat dan menjalankan *Virtual Machine*. Untuk *Hypervisors*, libvirt adalah sebuah *driver* untuk *Hypervisor* yang memungkinkan *live migration* dari satu *node* ke *node* lain.
- Nova-compute, bagian utama yang menjalankan operasi.
- Cinder *Block Storage* menyimpan semua data yang digunakan oleh *virtual machine*.

2.3. IPv4 Network dan Subnet Mask

Fungsi utama dari IP adalah menyediakan *logical address* untuk *host*. Sebuah IP *address* menyediakan struktur hirarkis baik untuk mengidentifikasi *host* secara unik maupun *network* dimana *host* itu berada [1]. Sebuah IP *address* paling sering dituliskan dengan desimal dalam format "x.x.x.x" dengan nilai x maksimal adalah 255. Setiap oktet yang terpisah oleh titik (.) merupakan angka 8-bit, sehingga IP *address* adalah 32-bit.

Subnet mask digunakan untuk menentukan besar dari sebuah *network*. Dalam bentuk desimal *subnet mask* tertulis dengan format yang sama dengan IP *Address*. Dalam bentuk bit, *subnet mask* dengan nilai 1 menetukan porsi *network* dan nilai 0 mentukan porsi *host*.

Porsi *network* (nilai 1) dari *subnet mask* harus berurutan sehingga *network address* untuk 158.80.0.0 dengan *subnet mask* 255.255.0.0 dapat dituliskan sebagai 158.80.0.0/16.

2.4. Aplikasi dan Saling Ketergantungan

Aplikasi paralel dibagi menjadi 2 sifat yaitu *loosely coupling* dan *tightly coupling*. Aplikasi *loosely coupling* adalah aplikasi paralel yang memiliki ketergantungan kecil antara proses-prosesnya sedangkan aplikasi *tightly coupling* adalah aplikasi paralel yang memiliki lebih banyak ketergantungan antara setiap proses-prosesnya.

Semakin *tight* sifat *coupling* aplikasi maka semakin banyak penggunaan *network* pada saat aplikasi dieksekusi disebabkan oleh perlunya aplikasi untuk mengakses data dari operasi *node* yang lain. Akses dalam hal ini mencakup menambahkan, mengubah dan menghapus data pada *node* lain.

Sebaliknya sifat *loose* cenderung memisahkan data menjadi bagian-bagian yang lebih kecil dan memproses data-data tersebut secara terpisah sehingga setiap *node* hanya mengoperasikan data yang menjadi bagiannya.

2.5. OpenStack *Identity Service* / Keystone

Keystone menyediakan *identity service* dan *access policy services* kepada semua komponen dalam OpenStack. Keystone mengimplementasikan API berbasis REST yaitu *Identity* API.

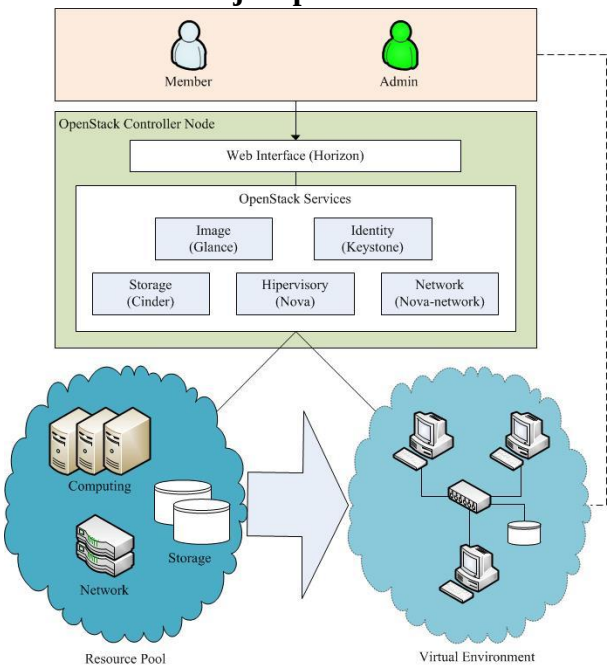
Keystone menyediakan autentikasi dan autorisasi untuk semua komponen OpenStack. Autentikasi memverifikasi bahwa permintaan sebenarnya berasal dari sumber yang benar, terdapat dua cara autentikasi yaitu berbasis *username*/*password* dan berbasis *Token*. Autorisasi memverifikasi hak akses *user* ke layanan yang diminta [5].

Komponen yang ada di dalam Keystone terdiri dari *tenant*, *service endpoint*, *regoin*, *user* dan *role* [5]. Berikut adalah penjelasan komponen-komponen tersebut:

- ***Endpoints***.
  Setiap *service* pada OpenStack memiliki sebuah URL (*host*) dan pada *port* khusus. URL dan *port* tersebut disebut *Endpoint*.
- ***Regions***.
  Region mendefinisikan lokasi fisik di dalam sebuah *data center*. Dalam pengaturan *cloud*, umumnya hampir semua *services* tersebar ke seluruh bagian *data center* atau *server* yang juga disebut *region*.
- ***User***.
  Sebuah akun yang terautentikasi oleh Keystone.
- ***Services***.
  Semua *service* yang terhubung dengan keystone adalah keystone *service*. Sebagai contoh, glance (*image service*) adalah keystone *service*.
- ***Role***
  Role membatasi akses *user* dalam infrastruktur *cloud*.
- ***Tenant***.
  Tenant adalah sebuah proyek dengan semua *service*, *endpoint* dan *user* pada *tenant* tersebut.

3. RANCANGAN SISTEM

3.1. Skema Kerja OpenStack



Gambar 1. Skema kerja sistem Openstack.

User yang telah terdaftar, dapat mengakses fitur IaaS yang disediakan oleh sistem *cloud* melalui *user interface* berupa *website*. *User* dapat meminta sebuah *virtual machine* dengan memilih *flavor* (spesifikasi misalnya CPU, RAM dan kapasitas *disk*) dan *image* (sistem operasi).

Controller node, dimana *website* berada, akan melakukan *scheduling*, yaitu memilih IP *address* untuk *virtual machine* dan *compute node* yang memenuhi kriteria sesuai dengan kapasitas yang diminta oleh *cloud user*. Pada tahap ini berbagai *service* Openstack akan bekerja sama untuk memeriksa hak *user* atas *hardware*, *image*, *network* dan *storage* sebelum menjalankan sebuah *instance* (*virtual machine*).

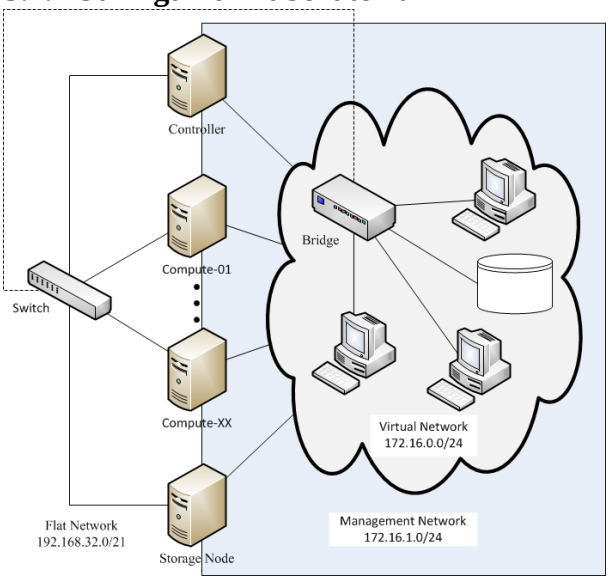
Sejumlah *instace* yang berjalan akan berada pada sebuah *virtual network* yang dibentuk pada *host* dimana *virtual machine* tersebut beroperasi. Hal ini seakan-akan bahwa sejumlah *instance* tersebut adalah sekumpulan *node* pada sebuah *cluster*.

User telah memiliki *virtual environmet* yang dibuat dari sekumpulan *virtual machine*. Setiap *virtual network* yang terbentuk terhubung dengan *network bridge* pada *physical interface* milik *host* sehingga dapat terhubung dengan *network* di luar *virtual network* tersebut, dalam hal ini *private network* milik Universitas Kristen Petra.

User sebagai pemilik *instance* harus mengakses *instance* tersebut melalui *console interface* yang disediakan pada *website* hanya pada saat pertama kali menjalankan *instance*. Langkah ini adalah tahap dimana *cloud user* melakukan pengaturan awal *sistem operasi* dan juga instalasi program-program lain bila diinginkan.

User dapat mengaktifkan koneksi pada *port-port* tertentu sesuai dengan *security group*, sejenis *firewall* milik OpenStack. Selanjutnya dapat dilakukan koneksi langsung ke *instance* tersebut tanpa melalui *interface web*.

3.2. Jaringan di Laboratorium



Gambar 2. Rancangan jaringan.

OpenStack membutuhkan 2 *network* yaitu *Management Network* dan *Public Network*. *Management Network* digunakan untuk menyediakan jalur komunikasi sistem dan administrasi *cloud*. *Public Network* menyediakan akses dari pengguna *cloud*, dalam kasus *private cloud*, *network* ini adalah *private network* (atau juga disebut *Flat Network*) milik Universitas Kristen Petra.

Flat network akan menggunakan pengaturan telah tersedia yang merupakan *private network* (192.168.32.0/248) milik Universitas Kristen Petra. Sedangkan untuk *Management Network*, dibuatkan sebuah *network* baru yang tidak terkait dengan *network* yang tersedia. *Network* 172.16.0.0/16 telah dipertimbangkan bersama dengan pihak yang berwenang dalam Universitas Kristen Petra yaitu Pusat Komputer (Puskom). Selanjutnya *network* tersebut akan dibagi, 172.16.1.0/24 menjadi *management network* dan sebagian untuk digunakan pada *virtual machine* yang akan beroperasi pada sistem *cloud*, adalah 172.16.0.0/24.

3.3. Lingkungan Dasar

Lingkungan dasar (*basic environment)* yang diperlukan adalah *node-node* dengan sistem operasi Ubuntu Server 14.04 LTS. Untuk setiap node, spesifikasi *hardware* sebagai berikut:

Processor	: Intel Core i5-3340 @ 3.1 GHz (2 *cores* / 4 *threads*)
RAM	: 16 GB
Disk	: 250 GB
Koneksi	: 1 *interface* 100Mbps *Ethernet*

Selain itu, dilakukan instalasi komponen-komponen dasar yang diperlukan oleh Openstack. Komponen-komponen dasar tersebut adalah sebagai berikut:

3.3.1. OpenStack Packages

Untuk instalasi OpenStack *packages*, Juno *cloud repository* harus ditambahkan pada *source-list* dari *Advanced Package Tool* (APT). Selanjutnya APT pada setiap *node* harus melakukan *update* dan *dist-upgrade*.

3.3.2. MariaDB

Database Management System (DBMS) bersifat wajib karena digunakan untuk menyimpan informasi untuk setiap *service*

yang berjalan pada OpenStack. Instalasi DBMS dilakukan pada *controller node*. MariaDB adalah sebuah DBMS berbasis *open source* yang disarankan dalam pembuatan *cloud* menggunakan OpenStack sejak versi Juno.

MariaBD memiliki banyak optimalisasi performa dibanding MySQL dalam melakukan eksekusi *query* sederhana. Sehingga MariaDB mampu menangani lebih banyak perintah SQL dalam satuan waktu.

3.3.3. RabbitMQ

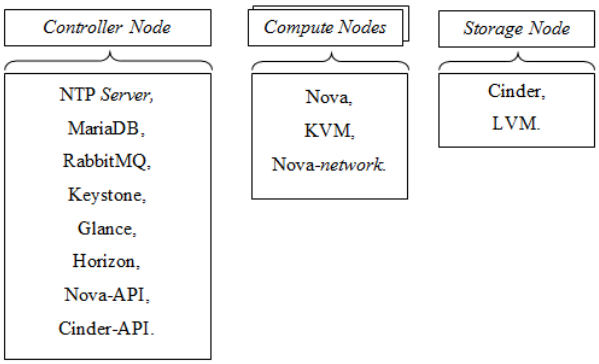
RabbitMQ adalah sebuah *message broker* yang menerima dan melanjutkan pesan. Instalasi RabbitMQ dilakukan pada *controller node*. Pada RabbitMQ, semua komponen OpenStack (*services*) akan bertindak sebagai *guest*. Pengaturan pada RabbitMQ tidak ada yang diubah kecuali *password* milik *guest*.

3.3.4. Network Time Protocol (NTP)

Instalasi NTP dilakukan ke seluruh *node* yang berada pada sistem *private cloud*. Meskipun demikian, peran *controller node* berbeda dengan *node-node* lainnya. NTP pada *controller node* memiliki fungsi untuk meyediakan pengaturan waktu kepada seluruh *node* lain dalam sistem *private cloud*. Untuk itu, pengaturan NTP pada *controller node*, harus menggunakan dirinya sendiri sebagai penyedia waktu dan menjawab permintaan NTP dari *node* lain.

Pada *node-node* selain *controller*, NTP akan diarahkan untuk menyesuaikan pengaturan waktu mereka dengan waktu yang disediakan oleh *controller node*. Sehingga seluruh *node* dalam sistem *private cloud* memiliki pengaturan waktu yang sama dengan *controller node*.

3.4. Framework Software



Gambar 3. Rancangan Nodes.

Framework yang digunakan adalah OpenStack versi Juno, maka *software* yang dimaksud adalah sejumlah komponen yang beroperasi pada sistem *cloud* berbasis OpenStack. Komponen-komponen tersebut terpisah menjadi beberapa *node* yaitu sebuah *controller node* dengan beberapa *compute node*, serta *node-node* tambahan seperti *storage node* dan *network node*.

3.4.1. Controller Node

Controller node adalah komputer yang bertugas untuk mengatur pengoperasian sistem secara keseluruhan. Pada *node* ini terdapat komponen-komponen sebagai berikut:

- Keystone
  Keystone sebagai *Identity service* harus mendaftarkan *service*, *admin* dan *user* yang akan menggunakan sistem.
- *Compute Management*
  Compute management adalah sebuah bagian dari nova *compute service* yang terletak pada *controller node*.
- Pengaturan Nova-*network*
  Pengaturan dilakukan pada *controller node* agar mengenal *legacy network* yang disediakan oleh *compute node*.
- Horizon
  Instalasi Horizon dilakukan setelah semua komponen utama telah berjalan dengan baik. Horizon merupakan *dashboard* yang digunakan oleh *user* untuk mengakses *service* yang disediakan oleh *private cloud*.
- Cinder API
  Cinder sebagai instalasi tambahan yang diperlukan untuk menyediakan *volume service*. *Service* ini diperlukan oleh *virtual machine* untuk memiliki media penyimpanan sehingga data masing-masing sesuai dengan keinginan pengguna.

  Instalasi yang dilakukan pada *controller node* hanyalah API dari cinder. Cinder API digunakan untuk melakukan pembagian kerja atas penggunaan *volume service* oleh semua *virtual machine* pada sistem.

3.4.2. Compute Node

Compute node bertindak sebagai *host* untuk *virtual machine* yang dijalankan oleh pengguna. Untuk itu, *compute node* memiliki *hypervisor* yaitu KVM, sebuah *virtual machine manager* (VMM) yang paling banyak digunakan dan telah teruji untuk sistem OpenStack.

Pengaturan lain yang dilakukan untuk membentuk nova-*network* yang melakukan pemberian IP *address* kepada *virtual machine*. *Nova* API terdapat pada *controller node* sehingga pengaturan dan pembagian *network* dan kerja tidak dilakukan pada *compute node*.

3.4.3. Storage Node

Storage node adalah sebuah *node* tambahan yang ditambahkan sebagai media penyimpanan yang digunakan sebagai *virtual hardisk* pada *virtual machine*. Sebelum pengaturan dilakukan perlu disiapkan sebuah partisi yang dibutuhkan sebagai media penyimpan. Partisi yang disediakan pada komputer milik laboratorium adalah 250 GB, sehingga partisi dibuat adalah 200 GB dengan menyediakan sisa 50 GB untuk sistem Ubuntu mencakup *root* dan *swap*.

Partisi tersebut dapat dibagikan kepada *user* dan berfungsi sebagai *disk* yang dapat dipindahkan antara *virtual machine*. Untuk penggunaannya, *user* perlu mengambil sebagian kapasitas untuk dijadikan sebuah *virtual disk*. Kemudian, *virtual disk* dapat di-*attach* ke sebuah VM dan *user* dapat mengakses *virtual disk* tersebut melalui NFS menggunakan VM dalam bentuk sebuah *physical disk*.

3.5. *Virtual Machine Image*

Private cloud yang dirancang harus memiliki *image* yang disediakan bagi pengguna. *Image* tersebut berisikan sistem operasi yang disediakan oleh pengembang sistem operasi tersebut dan telah diverifikasi oleh OpenStack sebagai sistem operasi yang layak beroperasi sebagai *virtual machine*. *Image-image* yang terpilih diantaranya adalah:

- Windows *Server* 2012 R2 *Standard Evaluation Edition*
- Ubuntu *Server* 14.04
- Debian (CLI)
- Windows XP 32-bit

3.6. *Virtual Machine Flavor*

Flavor adalah *template* yang disediakan sebagai spesifikasi *virtual machine. Flavor* mencakup CPU, RAM dan kapasitas *disk*. *Flavor-flavor* tersebut adalah sebagai berikut:

Tabel 1. Daftar *Flavor*.

Flavor	Jumlah *core* VCPU	Kapasitas RAM (GB)	Kapasitas Disk (GB)
Tiny	1	1	20
Small	1	2	40
Medium	2	4	60
Large	2	8	100
XLarge	4	14	200

3.7. *Website* untuk Registrasi *Member*

Website pendaftaran dibuat dengan menggunakan bahasa pemrograman PHP dan memanfaatkan fitur REST dari *framework Code Igniter*. Untuk mengakses data yang disediakan oleh *Identity Service* (Keystone), *website* akan mengakses API yang telah disediakan sehingga tidak langsung mengubah *database*.

4. IMPLEMENTASI SISTEM

4.1. Penempatan Peran Komputer di Laboratorium Komputer

Komputer pada laboratorium yang digunakan dimulai dengan MM-03 hingga MM-07 dengan jumlah 5 buah komputer. Peran MM-03 adalah *contoller node*, dimana komputer ini adalah *node* pertama dalam *network*. Selanjutnya dengan jumlah 3 komputer adalah *compute node*, peran tersebut dijalankan oleh MM-04, MM-05 dan MM-06. *Node* terakhir adalah MM-07 sebagai *storage node*. Selain itu, MM-11, MM-17, MM-18, MM-20 dan MM-21 ditambahkan sebagai *compute node* sehingga menjadi 8 *compute node*.

4.2. *Network*

Setiap *node* termasuk *controller node* membutuhkan 2 buah *network* yang terhubung sehingga selain dilakukan pengaturan, ditambahkan juga *virtual interface*. Pemberian *IP address* berurutan sesuai dengan nomor komputer di laboratorium. *Interface* utama diberikan IP address adalah 192.168.38.x dengan x adalah nomor komputer. *Virtual interface* diberikan *IP address* pada *network* 172.16.1.0/24 yaitu 172.16.1.x dengan x adalah nomor komputer.

4.3. Mendaftarkan *Host*

Untuk setiap *node* dapat mengenali *node* lain dalam *network* dan menggunakan *network* yang tepat maka setiap *node* harus dimasukan dalam daftar *IP address – Hostname pair* dalam *file* /etc/hosts pada setiap *node*.

4.4. Instalasi Openstack

Implementasi dilakukan dengan megikuti langkah-langkah yang disediakan pada buku instalasi yang telah disediakan oleh OpenStack. *Openstack Installation Manual* berisikan paduan proses instalasi yang dibutuhkan dalam implementasi yang dilakukan di laboratorium [7].

4.5. Memperbaiki *Bugs*

Dalam pengerjaan juga ditemukan beberapa masalah dalam implementasi. Masalah tersebut adalah sebagai berikut:

4.5.1. Service Nova

Service pada Nova memiliki ketergantungan terhadap *database* (MariaDB). Masalah ditemukan pada saat *controller node* (MM-03) di-*restart*. Masalah tersebut adalah bahwa *service* nova (termasuk nova-*cert*, nova-*consoleauth*, nova-*scheduler* dan nova-*conductor*) dijalankan terlebih dahulu oleh sistem operasi sebelum *service database* berhasil dijalankan. Hal ini mengakibatkan *service-service* nova tersebut mengalami kegagalan dan berhenti.

Solusi yang diambil adalah dengan mengubah *script* pada saat *initializing service* oleh sistem operasi (Ubuntu 14.04). *Script* tersebut berada pada direktori /etc/init/. Penambahan *script* akan membuat pemanggilan *service* nova menunggu *database* sebelum menjalankan *service* dari nova.

4.5.2. Error Cinder

Cinder atau *block storage* adalah untuk membagi sebuah partisi menjadi bagian-bagian kecil sesuai dengan keinginan pengguna dan memberikannya kepada pengguna. Pada saat permintaan disampaikan kepada cinder, aplikasi tidak dapat membagi partisi dan mengembalikan pesan *error*.

Hal ini terjadi karena aplikasi tidak memiliki hak akses untuk mengubah pengaturan *hardware*. Sehingga solusi yang diambil adalah dengan memberikan akses kepada cinder sebagai *super user*. Memberikan akses *super user* dapat dilakukan dengan mengedit *file* /etc/sudoers pada *node* MM-07. Berikut sebuah baris yang ditambahkan di akhir *file* tersebut:

```
cinder ALL=(ALL) NOPASSWD: ALL
```

Dengan melakukan hal ini, cinder telah dapat mengakses LVM dengan hak akses sebagai *super user* sehingga dapat melakukan partisi terhadap *disk*.

4.6. Menambahkan *Image*

Untuk mendapatkan *image*, dapat mengunduh *image* yang disediakan secara resmi atau membuat *image* tersebut dari *installer*. *Image* yang dapat didapat melalui *website* resmi adalah Windows *Server* 2012 R2 *Standard Evaluation Edition*, Ubuntu 14.04 dan Debian 8.0 (CLI). Sedangkan *image* Windows XP harus dibuat dengan Installer ISO.

Pembuatan Windows XP akan dilakukan komputer dengan GUI dan memanfaatkan salah satu *compute node*. Instalasi juga akan membutuhkan *floppy image* berisi SCSI *driver* virtio dan CD *image* berisi *driver* virtio untuk SCSI *Controller*, *Ethernet Controller*, dan VGA. *Floppy image* tersebut akan membantu saat instalasi *custom* SCSI *driver* sebelum *instalasi* Windows XP dan CD *image* setelah instalasi.

Semua *image* perlu di-*upload* ke glance agar dapat digunakan melalui sistem openstack. Berikut contoh perintah yang digunakan untuk meng-*upload image* Windows XP.

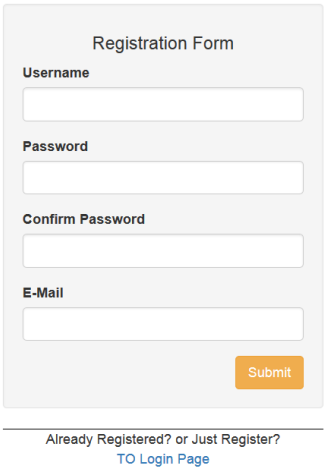
```
$ glance image-create --name "Windows XP" \
  --file wxp.qcow2 \
  --disk-format qcow2 --container-format bare \
  --is-public True --progress
```

4.7. Menambahkan *Website* untuk Registrasi *Member*

Website untuk registrasi *member* dibuat dengan bahasa pemrograman PHP dengan memanfaatkan fitur REST dari *framework Code Igniter*. Untuk akses API yang disediakan oleh keystone, pemrograman dibuat dengan memanfaatkan PHP-JSON dan PHP-CURL.

Akses API keystone dibuat dengan autentikasi sebagai admin sehingga registrasi seakan-akan dilakukan oleh admin.

Website menyediakan sebuah halaman untuk melakukan registrasi dan setelah selesai registrasi, *user* baru dapat mengakses *dashboard*.



Gambar 4. Tampilan Website Registrasi.

Website dibuat agar dapat memberikan notifikasi jika terjadi *error* pada *website* maupun *error* yang dikembalikan dari API.

4.8. Menjalankan *Instance*

Untuk mengetahui bahwa *user* dapat menggunakan fitur utama dari sistem IaaS OpenStack, percobaan untuk menjalankan *instance* dilakukan melaui *dashboard* Horizon. Tujuan dari hal ini adalah memastikan bahwa sistem dapat berjalan dan semua *image* yang disediakan dapat digunakan.

5. PENGUJIAN DAN EVALUASI

Pengujian dilakukan terhadap sistem dan komponen yang buat agar berjalan sesuai dengan yang diharapkan.

5.1. Fitur Base Image Cache

Pengujian melihat fungsi *base image caching* yang terjadi pada sistem, *node* yang telah memiliki *cache* untuk *base image* membutuhkan waktu untuk *spawning* selama kurang lebih 5 sampai 10 detik sedangkan untuk *node* yang tidak memiliki *cache* membutuhkan waktu pengiriman bergantung pada ukuran *image* yang digunakan. Berikut waktu yang dibutuhkan untuk menjalankan setiap *image* saat belum terdapat *cache*:

Tabel 2. *Image*, ukuran dan waktu *spawning* tanpa *cache*.

Nama *Image*	Ukuran *Image* (MB)	Waktu *Spawning*
Cirros-0.3.3-x86_64	13	10 – 15 detik
Ubuntu 14.04 *Trusty*	251	30 – 40 detik
Debbian Jessie 64-bit	447	40 – 60 detik
Windows XP	1.699	2 – 3 menit
Windows *Server* 2012	16.780	25 – 30 menit

5.2. Membanding Skenario Penempatan *Instance*

Penempatan *instance* (VM) dapat membentuk beberapa skenario pembagian *host*. Pengujian akan menggunakan hanya 2 buah skenario, yaitu sebagai berikut:

1. Skenario sebuah *instance* dalam sebuah *host*
2. Skenario banyak *instance* dalam sebuah *host*

Dengan menggunakan 2 buah skenario tersebut, pengujian dilakukan dengan menghitung waktu yang dibutuhkan oleh sistem untuk menjalankan sebuah instance hingga sistem operasi berjalan. Berikut hasil penggukuran waktu dengan menggunakan *stopwatch*:

Tabel 3 Waktu *boot image* skenario 1 dan skenario 2.

Image	Waktu dengan Skenario 1	Skenario 2	
		Jumlah Instance	Waktu
Debian Jessie	31 detik	4	1 menit
Windows XP	30 detik		90 detik
Ubuntu 14.04	3 menit		4 menit
Windows Server 2012	9 menit	2	11 menit

Hasil membuktikan bahwa jika banyak *instance* dijalankan pada sebuah *host* membutuhkan waktu yang lebih lama.

5.3. Sysbench

Sysbench adalah sebuah *benchmark tool* yang digunakan untuk menghitung kemampuan CPU, *memory* dan *file* I/O. Pengujian dilakukan dengan membuat sebuah *instance* yang paling mendekati dengan sebuah *node*, yaitu sebuah *instance* dengan menggunakan *flavor* terbesar (m1.xlarge).

Pengujian ditujukan untuk mengetahui seberapa dekat kemampuan yang dicapai oleh sebuah *instance* dengan kapabilitas sebuah *instance* dibandingkan dengan *host* yang sesungguhnya. Oleh karena itu, pengujian dilakukan terhadap sebuah *host* dan terhadap sebuah *instance*.

Dari hasil perhitungan dapat dikatakan bahwa penurunan terhadap kemampuan yang terbanyak adalah pada *memory* dan *hardisk*, sedangkan CPU tidak mengalami penurunan yang drastis.

5.4. HPCC *Benchmark*

HPCC (*High Performance Computer Challenge*) adalah sebuah *software benchmark* yang digunakan untuk mengukur performa dari sebuah sistem terdistribusi (*cluster*). HPCC adalah nama lain untuk LINPACK *Benchmark*, *software benchmark* yang disediakan oleh *top*500.org yang menyediakan statistik terkait *high-performance computer*.

Tabel 4. Hasil 12 kali benchmark yang dilakukan.

No. *Test*	# *Node*	*Core*/ *Node*	Hasil (Gflops)		*Overhead* (%)
			Instance	*Host* Fisik	
1	1	1	2.7670	2.7830	0.575%
2	1	2	4.6320	4.6600	0.601%
3	2	1	1.5180	1.5340	1.043%
4	1	4	7.6600	7.7780	1.517%
5	2	2	0.9660	1.0320	6.395%
6	4	1	0.8419	0.9171	8.199 %
7	2	4	1.1030	1.1720	5.887 %
8	4	2	0.7225	0.7478	3.383 %
9	8	1	0.7823	0.8512	8.094 %
10	4	4	0.5649	0.5951	5.074 %
11	8	2	0.5596	0.6031	7.212 %
12	8	4	0.5782	0.6130	5.676 %

Sebelum *benchmark* dilakukan, *host* yang akan digunakan harus dibuat untuk membentuk sebuah *cluster*. Salah satu hal yang harus dilakukan adalah membuat autentikasi SSH tanpa *password* dari satu *host* ke semua *host* lain dalam *cluster* yang

dibuat. Setelah *cluster* terbentuk, dilakukan instalasi HPCC dan pengujian siap dilakukan.

Pengujian dilakukan dengan cara membandingkan hasil *benchmark* menggunakan *instance* dengan hasil menggunakan *host* fisik. Hasil dari pengukuran dan perbandingan performa antara *instance* dan *host* fisik yang dihasilkan dengan menggunakan HPCC *Benchmark* dapat dilihat pada Tabel 4.

Dari data pada Tabel 4, dilihat bahwa performa sebuah *instance*/*virtual machine* sedikit lebih rendah terhadap *host* fisik. Di lain sisi, setiap hasil *benchmark* dengan menggunakan *multi-node* menghasilkan performa yang sangat rendah dibandingkan dengan *benchmark* pada sebuah *host*. Hal ini berarti bahwa penggunaan *muti-node* melalui *network* tidak meningkatkan performa aplikasi *tight-couple*.

5.5. Blender *Render*

Blender adalah *software* untuk melakukan *rendering* animasi dan dapat diatur menjadi sebuah *cluster render farm*. Blender adalah software yang bersifat *loose-couple*, memecahkan animasi menjadi beberapa *frame* yang dipisah proses *rendering*-nya pada banyak komputer.

Pengujian akan dilakukan dengan menggunakan animasi yang terdiri dari 50 *frame*. Jumlah *node* yang digunakan adalah 5 buah *virtual machine instance*. Pengaturan menggunakan 1 buah *node master* dan 4 buah *node slave*. Hasil dari pengujian yang dilakukan dapat dilihat pada Tabel 5.

Tabel 5. Hasil pengujian Blender *Render*.

No. Test	Jumlah *Node Slave*	Waktu *Rendering*
1	1	5 Jam 31 Menit 0 detik
2	2	2 Jam 54 Menit 34 detik
3	3	1 Jam 49 Menit 26 detik
4	4	1 Jam 19 Menit 16 detik

Kesimpulan yang dapat diambil adalah bahwa *private cloud* di laboratorium memiliki kecepatan *network* yang cukup untuk menunjang aplikasi terdistribusi yang bersifat *loose-couple*.

6. KESIMPULAN DAN SARAN

6.1. Kesimpulan

Dengan melakukan penelitian ini, dapat disimpulkan bahwa implementasi *private cloud* dengan memanfaatkan sumber daya komputer milik laboratorium memiliki hasil yang cukup memuaskan dalam beberapa hal, sebagai berikut:

- Fasilitas jaringan komputer yang tersedia mampu menopang performa dari sistem *private cloud* meskipun tetap memakan waktu yang lama dalam pengiriman image. Fitur *Image caching* cukup membantu dalam menyelesaikan masalah ini.
- Instance dalam sistem yang dihasilkan mampu mencapai performa yang mendekati sebuah *host* fisik sehingga kapabilitas CPU dapat dikatakan terpakai dengan maksimal.

Di lain sisi, implementasi *private cloud* ini juga memiliki kekurangan, yaitu:

- Kemampuan *network* yang tidak mendukung aplikasi seperti HPCC. Aplikasi *tight-coupling* ini mengalami penurunan performa secara drastis saat dijalankan dengan lebih dari sebuah komputer/*node*.

6.2. Saran

Peneliti mengharapkan bahwa pada penelitian berikutnya terhadap sistem *private cloud* di laboratorium komputer dapat membuat sistem yang lebih besar. *Private cloud* dapat dibuat dengan memanfaatkan keseluruhan laboratorium yang tersedia sehingga sistem memiliki sumber daya yang lebih banyak dan dapat melayani lebih banyak pengguna.

Peneliti juga mengharapkan untuk waktu ke depan, fasilitas *private cloud* dapat dimanfaatkan untuk kegiatan belajar mengajar. Hal ini dikarenakan kemampuan *cloud* yang mampu mencapai performa selayaknya sebuah *host* fisik.

Dikarenakan sistem mengalami penurunan akibat dari koneksi *network*, peneliti berharap sistem dapat beroperasi dengan fasilitas *network* yang lebih baik. Pada saat peneliti melakukan implementasi, sistem dibangun dengan fasilitas koneksi *network* berkecepatan 100Mbps. Sehingga diharapkan fasilitas *network* berikutnya dapat memiliki kecepatan lebih dari 100Mbps misalkan koneksi kabel *Fiber* berkecepatan 1Gbps.

7. REFERENSI

[1] Balchunas, A. 2013. *IPv4 Addressing and Subnetting v1.41*. Router Alley.

[2] Birke, R., Chen, L. Y., and Smirni, E. 2012. *Data Centers in the Wild: A Large Performance Study*. URI= http://domino.research.ibm.com/library/cyberdig.nsf/papers/0C306B31CF0D3861852579E40045F17F.

[3] Coulouris, G., Dollimore, J., Kindberg, and Blair, G. 2012. *Distributed Systems Concepts and Design*. Boston: Addison-Wesley.

[4] Glanz, J. 2012. *The Cloud Factories: Power, Pollution and the Internet*. URI= http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html.

[5] Jha, A., Johnson, D., Murari, K., Raju, M., Cherian, V., & Girikumar, Y. 2012. *OpenStack Beginner's Guide (for Ubuntu - Precise)*. CSS Corp. Pvt. Ltd.

[6] Mell, P., and Grance, T. 2011. *The NIST Definition of Cloud Computing*. Gaithersburg, United State: National Institute of Standards and Technology.

[7] OpenStack Foundation. 2014. *OpenStack Installation Guide for Ubuntu 14.04*. URI= http://docs.openstack.org/openstack-ops/openstack-ops-manual.pdf