BAB II

LANDASAN TEORI

2.1 Cloud Computing

Cloud computing adalah sebuah arsitektur teknologi informasi yang dimana sumber daya komputasi tersedia sebagai layanan yang dapat diakses melalui internet (Sasono: 2011).

Cloud computing pada dasarnya adalah menggunakan *internet-based service* untuk mendukung proses bisnis. *Cloud service* biasanya memiliki beberapa karakteristik, diantaranya adalah sangat cepat di *deploy*, sehingga cepat berarti *instant* untuk implementasi.

Teknologi *cloud* akan memberikan kontrak kepada *user* untuk *service* pada 3 tingkatan:

1. *Infrastructure as a service*, hal ini meliputi *grid* untuk *virtualized server, storage & network*. Contohnya seperti *amazon elastic compute cloud* dan *simple storage service.*
2. *Platform as a service*, hal ini memfokuskan pada aplikasi dimana dalam hal ini memungkinkan *developer* untuk tidak memikirkan *hardware* dan tetap fokus pada *application development*nya tanpa harus mengkhawatirkan *operating system, infrastructure scaling, load balancing* dan lainya. Contohnya yang telah mengimplementasikan ini adalah *force.com* dan *microsoft azure investment*.
3. *Software as a service*, hal ini memfokuskan pada aplikasi dengan *web-based interface* yang diakses melalui *web service* dan *web 2.0*. Contohnya adalah

google apps, *salesforce.com* dan *social network application* seperti *facebook.*

2.2 Platform as a Service (PaaS)

Platform as a Service (PaaS) adalah hasil pengembangan dari layanan *Software* as a Service (SaaS). Pada layanan SaaS, pengguna *cloud* hanya menggunakan software aplikasi pada sisi penggunaan saja, sedangkan PaaS adalah layanan dari *provider cloud* untuk digunakan oleh penggunanya dalam pembuatan sekaligus *hosting* aplikasi yang akan digunakan sebagai SaaS (Assagaf: 2011).

Dengan membuat (*developing program*) menggunakan layanan PaaS maka distribusi perangkat lunak hasil developing tersebut di tempatkan di (hosting) penyedia PaaS dan pembuatnya dapat mendistribusikan penggunaan fungsinya melalui internet, bahkan bisa mengkomersilkan layanan penggunaannya sebagai SaaS kepada pengguna lain (Assagaf: 2011).

Pada PaaS, kita membuat sendiri aplikasi *software* yang kita inginkan, termasuk skema *database* yang diperlukan. Skema itu kemudian kita pasang *(deploy)* di *server-server* milik penyedia jada PaaS. Penyedia jasa PaaS sendiri menyediakan layanan berupa *platform*, mulai dari mengatur *server-server* mereka secara virtualisasi sehingga sudah menjadi *cluster* sampai menyediakan sistem operasi di atasnya. Alhasil, kita sebagai pengguna hanya perlu memasang aplikasi yang kita buat di atasnya (Sasono: 2011).

2.3 Linux

Linux adalah nama yang diberikan kepada sistem operasi komputer bertipe *Unix*. *Linux* merupakan salah satu contoh hasil pengembangan perangkat lunak bebas dan sumber terbuka utama. Seperti perangkat lunak bebas dan sumber terbuka lainnya pada umumnya, kode sumber *Linux* dapat dimodifikasi, digunakan dan didistribusikan kembali secara bebas oleh siapa saja.

Nama "*Linux*" berasal dari nama pembuatnya, yang diperkenalkan tahun 1991 oleh Linus Torvalds. Sistemnya, peralatan sistem dan pustakanya umumnya berasal dari sistem operasi GNU, yang diumumkan tahun 1983 oleh Richard Stallman. Kontribusi GNU adalah dasar dari munculnya nama alternatif GNU/Linux.

Linux telah lama dikenal untuk penggunaannya di *server*, dan didukung oleh perusahaan-perusahaan komputer ternama seperti *Intel, Dell, Hewlett-Packard, IBM, Novell, Oracle Corporation, Red Hat, dan Sun Microsystems*. *Linux* digunakan sebagai sistem operasi di berbagai macam jenis perangkat keras komputer, termasuk komputer *desktop*, super komputer, dan sistem benam seperti pembaca buku elektronik, sistem permainan video (PlayStation 2, PlayStation 3 dan XBox), telepon genggam dan router. Para pengamat teknologi informatika beranggapan kesuksesan *Linux* dikarenakan *Linux* tidak bergantung kepada vendor (*vendor independence*), biaya operasional yang rendah, dan kompatibilitas yang tinggi dibandingkan versi *UNIX* tak bebas, serta faktor keamanan dan kestabilannya yang tinggi dibandingkan dengan sistem operasi lainnya seperti *Microsoft Windows*. Ciri-ciri ini juga menjadi bukti atas keunggulan model pengembangan perangkat lunak sumber terbuka (*opensource software*).

Sistem operasi *Linux* yang dikenal dengan istilah distribusi *Linux* (*Linux distribution*) atau distro *Linux* umumnya sudah termasuk perangkat-perangkat lunak pendukung seperti *server web*, bahasa pemrograman, basisdata, tampilan *desktop* (*desktop environment*) seperti GNOME, KDE dan Xfce juga memiliki paket aplikasi perkantoran (*office suite*) seperti OpenOffice.org, KOffice, Abiword, GNU. (Wikipedia, 2011)

2.4 *Ubuntu Enterprice Cloud* (UEC)

UEC adalah tumpukan aplikasi dari *Canonical* yang disertakan dengan *Ubuntu Server Edition*. UEC termasuk *Eucalyptus* bersama dengan sejumlah perangkat lunak yang *open source* lainnya. UEC membuatnya sangat mudah untuk menginstal dan menkonfigurasi *cloud*. *Canonical* juga menyediakan dukungan teknis komersial untuk UEC.

2.5 Eucalyptus

Eucalyptus adalah perangkat lunak yang tersedia di bawah GPL (*General Puclic Licence*) yang membantu dalam menciptakan dan mengelola *private cloud* atau bahkan dapat mengakses *cloud* secara publik. *Eucalyptus* telah menjadi sangat populer dan dipandang sebagai salah satu *open source platform cloud* utama.

Nama *Eucalyptus* adalah singkatan dan berdiri untuk *Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems* yang artinya arsitektur *utility computing* elastis untuk menghubungkan program anda

untuk sistem berguna. Sebuah deskripsi singkat dari periode ini dapat dibaca di sini.

Eucalyptus memungkinkan penciptaan di lingkungan awan pribadi, dengan tidak ada persyaratan untuk memperbarui peralatan infrastruktur organisasi TI yang ada atau perlu untuk memperkenalkan perangkat keras khusus.

2.5.1 Node Controller (NC)

Node Controller adalah sebuah VT (*Virtualization* Technology) server yang mampu menjalankan KVM (Kernel-based Virtual Machine) sebagai hypervisor. eucalyptus otomatis menginstal KVM ketika pengguna memilih untuk menginstal Node Controller. VM (Virtual Machine) berjalan pada hypervisor dan dikendalikan oleh eucalyptus yang disebut instance. Eucalyptus mendukung hypervisors lain seperti Xen selain KVM. Namun Canonical telah memilih KVM sebagai pilihan hypervisor untuk UEC.

Node Controller berjalan pada setiap node dan mengontrol siklus hidup (life cycle) dari instance yang berjalan pada node. Node Controller berinteraksi dengan sistem operasi dan hypervisor yang berjalan pada Node di satu sisi dan Cluster Controller di sisi lain.

Node Controller memerintahkan sistem operasi yang berjalan pada node untuk menemukan physical resources dari node yang berupa jumlah core, ukuran memori, ruang disk yang tersedia dan juga untuk mempelajari tentang keadaan VM instance yang berjalan pada node dan menyebarkan data ini sampai dengan CC.

Fungsi dari Node Controller adalah:

1. Pengumpulan data yang terkait dengan ketersediaan *resource* dan pemanfaatan pada *node* dan pelaporan data ke CC
2. Manajemen siklus hidup (*life cycle*) *instance*

2.5.2 Cluster Controller (CC)

Cluster Controller mengelola satu atau lebih Node Controller dan menyebarkan / mengelola contoh pada mereka. Cluster Controller juga mengelola jaringan untuk instance yang berjalan pada Nodes di bawah beberapa jenis mode jaringan dari Eucalyptus.

Cluster Controller berkomunikasi dengan Cloud Controller (CLC) di satu sisi dan Node Controller di sisi lain.

Fungsi dari Cluster Controller adalah:

1. Untuk menerima permintaan dari Cluster Controller untuk menyebarkan instance.
2. Untuk memutuskan mana Node Controller yang digunakan untuk menyebarkan instance
3. Untuk mengontrol jaringan virtual (virtual network) yang tersedia untuk instance.
4. Untuk mengumpulkan informasi tentang Node Controller yang terdaftar dengan instance dan melaporkannya kepada Cluster Controller

2.5.3 Walrus Storage Controller (WS3)

Walrus Storage Controller menyediakan layanan penyimpanan persisten sederhana dengan menggunakan REST (*Representational State Transfer*) dan SOAP (*Simple Object Access Protocol)* API yang kompatibel dengan API S3 (*Simple Storage Service*).

Fungsi dari *Walrus Storage Controller* adalah:

1. Menyimpan mesin image.
2. Menyimpan snapshot.
3. Menyimpan dan melayani file menggunakan S3 (*Simple Storage Service*) API

Walrus Storage Controller harus dianggap sebagai suatu sistem penyimpanan file sederhana.

2.5.4 Storage Controller (SC)

Storage Controller menyediakan penyimpanan blok persisten untuk digunakan oleh instance. Hal ini mirip dengan EBS (Storage Blok elastis) layanan dari AWS (Amazon Web Service).

Fungsi dari Storage Controller adalah:

1. Menciptakan perangkat EBS persisten.
2. Menyediakan penyimpanan blok atas AoE atau iSCSI (*Internet Small Computer System Interface*) protokol untuk *instance*.
3. Memungkinkan penciptaan *snapshot* dari *volume*.

2.5.5 *Cloud Controller* (CLC)

Cloud Controller adalah *front end* untuk infrastruktur *cloud* keseluruhan. *Cloud Controller* menyediakan antarmuka layanan *web* ke *client tool* di satu sisi dan berinteraksi dengan seluruh komponen infrastruktur *Eucalyptus* di sisi lain. *Cloud Controller* juga menyediakan antarmuka *web* untuk pengguna guna mengelola aspek-aspek tertentu dari infrastruktur UEC.

Fungsi dari *Cloud Controller* adalah:

1. Memonitor ketersediaan *resource* pada berbagai komponen infrastruktur *cloud*, termasuk *hypervisor node* yang digunakan untuk benar-benar ketentuan *instance* dan pengendali *cluster* yang mengelola *node hypervisor*.
2. *Resource arbitrase*

   Memutuskan mana *cluster* yang akan digunakan untuk pengadaan *instance*
3. Pemantauan *instance* yang berjalan

Singkatnya *Cloud Controller* memiliki pengetahuan yang komprehensif tentang ketersediaan dan penggunaan *resource* di *cloud* dan keadaan *cloud*.

2.6 OpenStack

OpenStack adalah teknologi *cloud computing* yang menyediakan sistem operasi *cloud* untuk *public* dan *private cloud* di bawah *Apache License*. Saat ini telah didukung oleh lebih dari 60 *company* yang berkontribusi untuk mengembangkan teknologi ini.

OpenStack ini adalah sebuah kolaborasi global pengembang dan teknologi komputasi awan yang memproduksi *platform cloud computing* untuk *public* dan *private cloud*.

Proyek ini bertujuan untuk memberikan solusi ke semua jenis awan dengan menjadi sederhana untuk diimplementasikan, kebutuhan skala besar, dan banyak fitur.

Teknologi ini terdiri dari serangkaian proyek yang saling terkait dengan memberikan berbagai komponen untuk solusi infrastruktur awan.

2.6.1 *OpenStack Compute Service* (*Nova*)

Open souce Software yang di desain untuk me-*manage* jaringan-jaringan skala besar, *virtual* mesin serta menciptakan *platform* yang *scalable* untuk *cloud-computing*.

Computing Fabric controller Semua kegiatan yang diperlukan untuk mendukung siklus hidup dari *instance* dalam *OpenStack cloud* yang ditangani oleh *Nova*. Hal ini membuat *Nova* sebagai *Platform* Manajemen yang mengelola sumber daya komputasi, jaringan, otorisasi, dan kebutuhan skalabilitas dari *OpenStack cloud*. Namun, *Nova* tidak menyediakan kemampuan *virtualisasi* dengan sendirinya, melainkan menggunakan *API Libvirt* untuk berinteraksi dengan didukung *hypervisors*. Nova memperlihatkan semua kemampuannya melalui *API* layanan *web* yang kompatibel dengan *API EC2* dari *Amazon Web Services*.

Fungsi dan Fitur dari *OpenStack Compute* adalah sebagai berikut:

1. Mengelola siklus hidup (*life cycle*) dari *instance*.
2. Mengelola *compute resource*
3. Jaringan dan Otorisasi
4. REST (*Representational State Transfer*) berbasis API

5. Komunikasi *asynchronous* yang konsisten
6. *Hypervisor* agnostik: dukungan untuk *Xen, XenServer / XCP, KVM, UML, VMware vSphere dan Hyper-V*

Komponen utama dari *OpenStack Compute* adalah sebagai berikut:

1. API *Server* (*nova-api*)

Server API menyediakan sebuah *interface* bagi dunia luar untuk berinteraksi dengan infrastruktur *cloud*. API *server* adalah satu-satunya komponen yang memungkinkan dunia luar untuk mengelola infrastruktur. Manajemen dapat melakukan layanan *web* melalui panggilan menggunakan EC2 API. *Server* API kemudian dapat berkomunikasi dengan komponen yang relevan dari infrastruktur *cloud* melalui Antrian Pesan (*Message Queue*). Sebagai alternatif untuk EC2 API, *OpenStack* juga menyediakan API asli yang disebut "*OpenStack* API".

2. *Message Queue* (*rabbit-mq server*)

OpenStack berkomunikasi di antara mereka sendiri menggunakan antrian pesan (*Message Queue*) melalui AMQP (*Advanced Message Queue Protocol*). *Nova* menggunakan panggilan *asynchronous* untuk merespon permintaan, dengan panggilan yang kembali akan memicu sekali respon diterima. Karena komunikasi *asynchronous* digunakan, pengguna tidak akan terlalu lama dalam keadaan menunggu. Ini berlaku efektif sejak banyak tindakan yang diharapkan oleh *API* panggilan seperti peluncuran sebuah contoh atau meng-*upload* gambar yang memakan waktu.

3. *Compute Workers* (*nova-compute*)

Compute workers berurusan dengan siklus hidup manajemen *instance*. Mereka menerima permintaan untuk manajemen siklus hidup *instance* melalui

Message Queue dan melaksanakan operasi. Ada beberapa *Compute workers* dalam penyebaran awan produksi yang khas. *instance* ditempatkan pada salah satu *Compute workers* tersedia berdasarkan algoritma penjadwalan yang digunakan.

4. *Network Controller* (*nova-network*)

Network controller berkaitan dengan konfigurasi jaringan dari mesin *host*. Ia melakukan operasi seperti mengalokasikan alamat *IP*, mengkonfigurasi *VLAN* untuk proyek-proyek, pelaksanaan *security groups* dan mengkonfigurasi jaringan untuk *compute nodes*.

5. *Volume Worker* (*nova-volume*)

Volume worker digunakan untuk pengelolaan *volume instance* berbasis LVM (*Logical Volume Manager*). *Volume worker* melakukan fungsi *volume* yang terkait seperti penciptaan, penghapusan, melampirkan *volume* ke *instance*, dan memisahkan *volume* dari *instance*. *Volume* menyediakan cara untuk menyediakan penyimpanan persisten untuk *instance*, sebagai partisi *root* adalah non-persisten dan setiap perubahan yang dibuat itu akan hilang ketika *instance* dihentikan. Ketika *volume* terlepas dari *instance* atau ketika sebuah contoh, dimana *volume* terpasang, dihentikan, itu mempertahankan data yang tersimpan di dalamnya. Data ini dapat diakses dengan kembali melampirkan *volume* ke *instance* yang sama atau dengan melampirkan ke *instance* lainnya.

Data penting dalam *instance* harus selalu ditulis untuk *volume*, sehingga dapat diakses nantinya. Hal ini biasanya berlaku untuk kebutuhan penyimpanan *database server* dll.

6. Scheduler (nova-scheduler)

The scheduler memetakan *nova*-API yang dipanggi untuk komponen *OpenStack* yang sesuai. Ini berjalan sebagai daemon bernama *nova-scheduler* dan mengambil *compute server* dari sumber daya yang tersedia tergantung pada algoritma penjadwalan di tempat. *Scheduler* dapat mendasarkan keputusan pada berbagai faktor seperti beban, memori, jarak fisik dari zona ketersediaan (*availability zone*), CPU arsitektur, dll. *nova scheduler* menerapkan arsitektur *pluggable*.

Saat ini *nova-scheduler* mengimplementasikan algoritma penjadwalan beberapa dasar:

a. *Change*: Dalam metode ini, compute *host* dipilih secara acak di seluruh *availability zone*.

b. *Availability zone*: Serupa dengan *change*, tapi *compute host* tersebut dipilih secara acak dari dalam *Availability zone* tertentu.

c. *Simple*: Dalam metode ini, *host* memiliki beban yang paling sehingga dipilih untuk menjalankan *instance*. Informasi beban dapat diambil dari penyeimbang beban (*load balancer*).

2.6.2 *OpenStack Imaging Service* (*Glance*)

OpenStack Imaging Service adalah sebuah pencarian dan pengambilan sistem untuk mesin virtual gambar (*virtual machine images*). Hal ini dapat dikonfigurasi untuk menggunakan salah satu dari *backends* penyimpanan (*storage backends*) berikut:

1. *Local filesystem* (default)

2. *OpenStack Object Store* untuk menyimpan gambar
3. S3 (*Simple Storage Service*) penyimpanan langsung
4. S3 (*Simple Storage Service*) penyimpanan dengan obyek penyimpanan sebagai perantara untuk akses S3.
5. *HTTP* (*read-only*)

OpenStack Imaging Service adalah salah satu produk dari *OpenStack* yang digunakan untuk layanan *virtual disk images*.

Fungsi dan Fitur dari *OpenStack Compute* adalah sebagai berikut:

1. Menyediakan layanan *image*.

Kompunen utama dari *OpenStack Compute* adalah sebagai berikut:

1. *Glance-control*
2. *Glance-registry*

2.6.3 *OpenStack Storage Service* (*Swift*)

Swift mampu menyimpan miliaran objek yang didistribusikan di seluruh *node*. *Swift* sudah *built-in* sistem *redundansi* dan manajemen *failover* dan mempunyai kemampuan untuk mengarsipkan dan media *streaming*. Hal ini sangat *scalable* baik dari segi ukuran (*several petabyte*) dan kapasitas (*number of objects*).

Fungsi dan Fitur dari *Swift* adalah sebagai berikut:

1. Penyimpanan objek berjumlah besar (banyak).
2. Penyimpanan objek berukuran besar.
3. Data Redundansi.
4. Arsip kemampuan

Bekerja dengan *dataset* besar.

5. Data *container* untuk *mesin* virtual dan aplikasi *cloud*.
6. Kemampuan media *streaming*.
7. Keamanan penyimpanan objek.
8. *Backup* dan arsip.
9. Skalabilitas yang ektrim.

Komponen utama dari *Swift* adalah sebagai berikut:

1. *Swift Proxy Server*

Para konsumen berinteraksi dengan *Swift setup* melalui *proxy server* yang menggunakan *API Swift*. *Proxy server* bertindak sebagai *gatekeeper* dan menerima permintaan dari dunia. *Proxy server* akan melihat lokasi yang tepat dan me-*route* permintaan kepada mereka.

Proxy server juga menangani kegagalan dari entitas dengan me-*routing* permintaan kembali untuk entitas *failover* (*handoff entities*).

2. *Swift Object Server*

Object server bertanggung jawab untuk menangani penyimpanan, pengambilan dan penghapusan objek yang tersimpan dalam penyimpanan lokal. Obyek biasanya berupa *file biner* yang disimpan dalam *filesystem* dengan *metadata* yang terkandung sebagai atribut *file* yang diperpanjang (*xattr*).

3. *Swift Container Server*

Container server mendaftarkan objek dalam sebuah *container*. Daftar yang disimpan dijadikan sebagai *file SQLite*. *Container server* juga melacak statistik seperti jumlah objek yang terkandung dan ukuran penyimpanan yang ditempati oleh *container* tersebut.

4. *Swift Account Server*

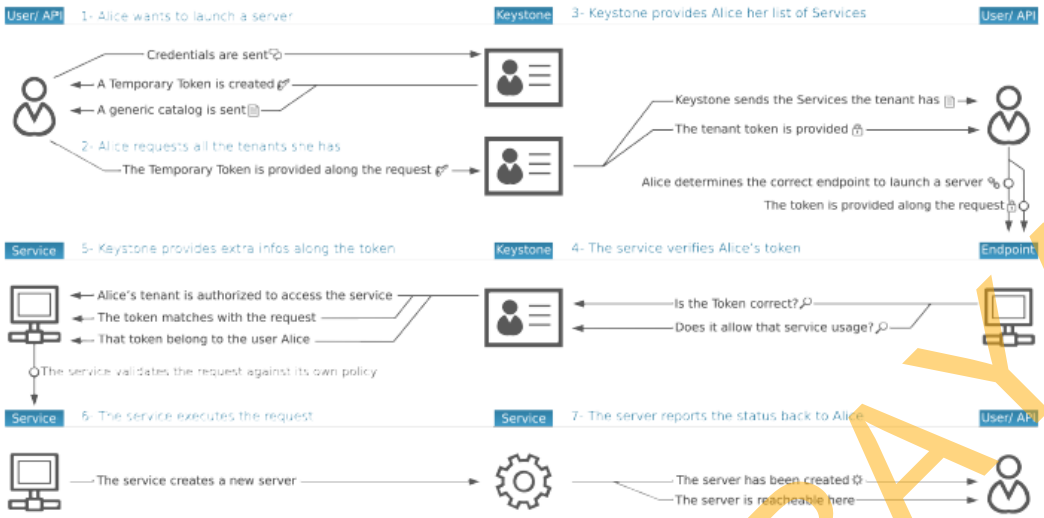
   Account server mendaftarkan *container* sama seperti *container server* mendaftarkan objek.

5. The RING

   RING berisi informasi tentang lokasi fisik dari objek-objek yang tersimpan di dalam Swift. Ini adalah representasi virtual pemetaan nama entitas untuk lokasi nyata fisik mereka. Hal ini sejalan dengan layanan pengindeksan yang digunakan untuk berbagai proses pencarian dan menemukan lokasi fisik sebenarnya dari entitas dalam cluster.

2.6.4 *OpenStack Identity Service* (*Keystone*)

Keystone menyediakan layanan identitas dan akses kebijakan untuk semua komponen dalam keluarga *OpenStack*. *Keystone* menerapkan itu di *REST*-nya sendiri yang berbasis *API* (*Identity API*). *Keystone* menyediakan otentikasi dan otorisasi untuk semua komponen *OpenStack* termasuk (namun tidak terbatas pada) *Swift*, *Glance*, *Nova*. Otentikasi memverifikasi bahwa sebuah permintaan itu sebenarnya berasal dari siapa yang mengakatakan itu tidak. Otorisasi akan memverifikasi apakah pengguna yang terotentikasi memiliki akses ke layanannya yang dia minta. Pada Gambar 2.1 dapat dilihat *Keystone Manager Indentity*.



Gambar 2.1 *Keystone Identity Manager*

Keystone menyediakan dua cara otentikasi. Salah satunya adalah *username* / *password* dan yang lainnya adalah *token*. Selain itu, *keystone* menyediakan layanan-layanan berikut:

1. *Token Service* (yang membawa informasi mengenai otorisasi pengguna yang terotentikasi).
2. *Catalog Service* (yang berisi daftar layanan yang tersedia di *users disposal*).
3. *Policy Service* (*keystone* mengelola akses ke layanan tertentu oleh pengguna atau kelompok tertentu).

Komponen utama dari *identity service* adalah sebagai berikut:

1. *Endpoint*

Setiap layanan *openstack* (*Nova*, *Swift*, *Glance*) berjalan pada *port* khusus dan pada *URL* khusus (*host*).

2. *Regions*

Sebuah daerah yang mendefinisikan lokasi fisik khusus di dalam pusat data. Dalam mengkonfigurasi *cloud*, sebagian besar, jika tidak semua layanan didistribusikan di seluruh pusat data / *server*.

3. *User*

Sebuah *keystonte* telah diotentikasi oleh *user*.

4. *Services*

Setiap komponen yang sedang terhubung ke atau yang diberikan melalui *keystone* bisa disebut layanan. Sebagai contoh, kita dapat memanggil *Glance* melalui layanan *keystone*.

5. *Role*

Dalam rangka mempertahankan batasan seperti pengguna tertentu dapat melakukan sesuatu di dalam infrastruktur *cloud*, adalah penting untuk memiliki peran terkait.

6. *Tenant*

Tenant adalah sebuah proyek dengan semua layanan *endpoint* dan *role* yang berhubungan dengan pengguna yang merupakan anggota dari bahwa *tenant* tertentu

2.6.5 OpenStack UI Service (Horizon)

Horizon dashboard berbasis *web* dapat digunakan untuk mengelola / mengatur layanan *OpenStack*. Hal ini dapat digunakan untuk mengelola *instances* dan *images*, membuat *keypairs*, melampirkan *volume* ke *instance*, memanipulasi *Swift container* dll. Selain itu, *dashboard* bahkan memberikan akses pengguna ke konsol *instance* dan dapat terhubung ke sebuah *instance* melalui *VNC*. Secara keseluruhan, *Horizon* fitur berikut:

1. *Instance Management*

Membuat atau menghentikan *instance*, melihat *log* dari konsol dan terhubung melalui *VNC*, Melampirkan *volume*, dll.

2. *Access and Security Management*

   Membuat *security groups*, mengelola *keypairs*, menetapkan *floating IP*, dll.

3. *Flavor Management*

   Kelola *flavor* yang berbeda atau *instance virtual hardware templates*.

4. *Image Management*

   Meng*edit* atau menghapus *images*.

5. Lihat katalog layanan.

6. Mengelola *user*, kuota dan penggunaan untuk proyek-proyek.

7. *User Management*

   Membuat *user*, dll.

8. *Volume Management*

   Menciptakan *Volume* dan *snapshot*.

9. *Object Store Manipulation*

   Membuat, menghapus *container* dan objek.

10. Men-*download* variabel lingkungan untuk sebuah proyek.

2.7 Phoronix Test Suite

Phoronix Test Suite adalah pengujian paling komprehensif dan platform lain dalam hal ini menyediakan kerangka *extensible* untuk test. Perangkat lunak ini juga di rancang dengan tolak ukur kualitatif dan kuantitatif dengan mengukur secara bersih dan mudah di gunakan.

Phoronix Test Suite didasarkan pada pengujian ekstensif dan alat internal yang dikembangkan oleh *Phoronix.com* sejak tahun 2004 bersama dengan dukungan dari terkemuka tier-satu perangkat keras komputer dan vendor

perangkat lunak. Perangkat lunak ini open source dan berlisensi di bawah GPLv3 GNU.

Awalnya dikembangkan untuk pengujian *Linux* otomatis, mendukung ke *Phoronix Test Suite* sejak itu telah ditambahkan untuk *OpenSolaris*, *Apple Mac OS X*, *Microsoft Windows*, dan sistem operasi BSD. *Phoronix Test Suite* terdiri dari inti pengolahan ringan *(Poin-core)* dengan masing-masing tolok ukur yang terdiri dari profil berbasis XML dan *script* sumber daya terkait.

1. OpenBenchmarking.org Integration
2. 130+ Test Profiles
3. 60+ Test Suites
4. Extensible Testing Architecture
5. Optional Linux-based LiveDVD/USB Testing Distribution (PTS Desktop Live)
6. Automated Test Installation
7. Dependency Management Support
8. Module-based Plug-In Architecture
9. PNG, JPG, GIF, Adobe SWF, SVG Graph Rendering Support
10. Automated Batch Mode Support
11. Global Database For Result Uploads, Benchmark Comparisons
12. Detailed Software, Hardware Detection
13. System Monitoring Support
14. GTK2 Graphical User Interface + Command-Line Interface
15. Runs On Linux, OpenSolaris, Mac OS X, Windows 7, & BSD Operating Systems

Fitur Phoronix Test Suite antara lain:

1. Mudah di gunakan
2. Arsitektur Extensible
3. Statistik Akurasi
4. Perekaman Hasil
5. Multi-Platform

2.8 Uji Statistika

Untuk menguji hasil dari *benchmarking instance*, penulis menggunakan pengujian hipotesis karena pengujian ini didasarkan atas analisa data dan populasi data.

Pada pengujian hipotesis ini penulis menggunakan pengujian rata-rata antara hasil *benchmarking* dari *instance Eucalyptus* dan *instance* dari *OpenStack*. Namun sebelum uji rataan dapat dilakukan, uji varian harus dilakukan terlebih dahulu untuk mengetahui sifat dari data uji homogen atau heterogen sesuai dengan Tabel 2.1.

Tabel 2.1 Uji Variansi

H_0	Uji Statistik	H_1	Daerah Kritis
$\sigma_1^2 = \sigma_2^2$	$F = \frac{S_1^2}{S_2^2}$ $v_1 = n_1 - 1$ dan $v_2 = n_2 - 1$	$\sigma_1^2 < \sigma_2^2$ $\sigma_1^2 > \sigma_2^2$ $\sigma_1^2 \neq \sigma_2^2$	$t < -t_\alpha$ $t > t_\alpha$ $t < -t_{\alpha/2}$ dan $t > t_{\alpha/2}$

Apabila berdasarkan uji variansi ternyata nilai H_0 diterima, maka hal ini berarti bahwa data tersebut bersifat homogen. Maka uji rata-rata dilakukan dengan menggunakan rumus sesuai Tabel 2.2

Tabel 2.2 Hipotesis Uji Rata-Rata Bila Data Homogen

H_0	Uji Statistik	H_1	Daerah Kritis
$\mu_1 - \mu_2 = d_0$	$T' = \frac{\left(\bar{X}_1 - \bar{X}_2\right) - d_0}{S_p\sqrt{1/n_1 + 1/n_2}}$ $S_p^2 = \frac{(n_1-1)S_1^2 + (n_2-1)S_2^2}{n_1+n_2-2}$ $v = n_1 + n_2 - 2$ $\sigma_1 \neq \sigma_2$ dan tidak diketahui	$\mu_1 - \mu_2 < d_0$ $\mu_1 - \mu_2 > d_0$ $\mu_1 - \mu_2 \neq d_0$	$T' < -t_{\alpha,v}$ $T' > t_{\alpha,v}$ $T' < -t_{\alpha/2,v}$ dan $T' > t_{\alpha/2,v}$

Apabila berdasarkan uji variansi ternyata nilai H_0 ditolak, maka hal ini berarti bahwa data tersebut bersifat heterogen. Maka uji rata-rata dilakukan dengan menggunakan rumus sesuai Tabel 2.3:

Tabel 2.3 Hipotesis Uji Rata-Rata Bila Data Heterogen

H_0	Uji Statistik	H_1	Daerah Kritis
$\mu_1 - \mu_2 = d_0$	$T' = \frac{\left(\bar{X}_1 - \bar{X}_2\right) - d_0}{\sqrt{S_1^2/n_1 + S_2^2/n_2}}$ $v = \frac{(S_1^2/n_1 + S_2^2/n_2)^2}{\frac{(S_1^2/n_2)^2}{n_1-1} + \frac{(S_2^2/n_2)^2}{n_2-1}}$ $\sigma_1 \neq \sigma_2$ dan tidak diketahui	$\mu_1 - \mu_2 < d_0$ $\mu_1 - \mu_2 > d_0$ $\mu_1 - \mu_2 \neq d_0$	$T' < -t_{\alpha,v}$ $T' > t_{\alpha,v}$ $T' < -t_{\alpha/2,v}$ dan $T' > t_{\alpha/2,v}$