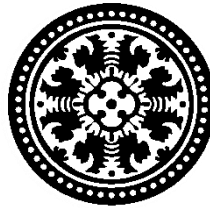


**PROPOSAL TUGAS AKHIR**

**SISTEM MANAJEMEN LAYANAN WEB BERBASIS PLATFORM AS A  
SERVICE (PAAS) DENGAN API OPENSTACK**

**KOMPETENSI JARINGAN KOMPUTER**



**IDA BAGUS RATHU EKA SURYA WIBAWA**  
**NIM. 1308605045**

**JURUSAN ILMU KOMPUTER**  
**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**  
**UNIVERSITAS UDAYANA**  
**BUKIT JIMBARAN**  
**2017**

## LEMBAR PENGESAHAN TUGAS AKHIR

Judul : Sistem Manajemen Layanan *Web Berbasis Platform as a Service (PaaS)* dengan *API Openstack*  
Kompetensi : Jaringan Komputer  
Nama : Ida Bagus Rathu Eka Surya Wibawa  
NIM : 1308605045  
Tanggal Seminar : 6 Oktober 2017

Disetujui oleh:

Pembimbing I

Penguji I

I Dewa Made Bayu Atmaja Darmawan, S.Kom., M.Cs.  
NIP. 198901272012121001

I Gede Santi Astawa, S.T., M.Cs.  
NIP. 198012062006041003

Pembimbing II

Penguji II

I Gede Oka Gartria Atitama, S.Kom., M.Kom.  
NIP. 1991022620160312001

Gst. Ayu Vida Mastrika Giri, S.Kom., M.Cs.  
NIP. 1990060620160322001

Penguji III

Made Agung Raharja, S.Si., M.Cs.  
NIK. 1985091920130122003

Mengetahui,  
Ketua Komisi Seminar dan Tugas Akhir  
Jurusan Ilmu Komputer FMIPA Universitas Udayana

I Gede Santi Astawa, S.T., M.Cs.  
NIP. 198012062006041003

## KATA PENGANTAR

Puji syukur peneliti panjatkan kehadapan Tuhan Yang Maha Esa, karena berkat rahmat dan karunia-Nya, Proposal Tugas Akhir dengan judul “Sistem Manajemen Layanan *Web Berbasis Platform as a Service (PaaS)* dengan *API Openstack*” dapat diselesaikan dengan baik dan tepat waktu. Proposal ini diharapkan dapat menjadi pedoman dan arahan dalam melaksanakan penelitian. Peneliti mengucapkan terima kasih kepada berbagai pihak yang telah membantu dalam pembuatan proposal ini, antara lain:

1. Bapak I Dewa Made Bayu Atmaja Darmawan, S.Kom. M.Cs. sebagai pembimbing 1 yang telah mengkritisi, membimbing, dan menyempurnakan proposal tugas akhir ini.
2. Bapak I Gede Oka Gartria, S.Kom., M.Kom. sebagai pembimbing 2 yang telah membimbing, dan menyempurnakan proposa tugas akhir ini.
3. Bapak Agus Muliantara, S.Kom., M.Kom. selaku Ketua Jurusan Ilmu Komputer yang telah banyak memberikan masukan dalam proses pembuatan proposal tugas akhir ini.
4. Bapak/Ibu dosen di Jurusan Ilmu Komputer, yang telah meluangkan waktu turut memberikan saran dan masukan dalam penyempurnaan proposal tugas akhir ini.
5. Semua pihak yang telah memberi dukungan sehingga proposal ini dapat diselesaikan sesuai dengan waktu yang ditentukan.

Disadari pula bahwa sudah tentu laporan ini masih mengandung kelemahan dan kekurangan. Maka dari pada itu masukan dan saran penyempurnaan sangat diharapkan.

Bukit Jimbaran, Oktober 2017

Penyusun

Ida Bagus Rathu Eka Surya Wibawa

## DAFTAR ISI

PROPOSAL TUGAS AKHIR .....	i
LEMBAR PENGESAHAN TUGAS AKHIR .....	ii
KATA PENGANTAR .....	iii
DAFTAR ISI.....	iv
DAFTAR GAMBAR .....	vi
DAFTAR TABEL.....	vii
1. Latar Belakang .....	1
2. Rumusan Masalah.....	2
3. Tujuan Penelitian .....	2
4. Batasan Masalah .....	3
5. Manfaat Penelitian .....	3
6. Tinjauan Pustaka.....	3
6.1. Tinjauan Studi .....	3
6.2. <i>Cloud Computing</i> .....	7
6.2.1. <i>Jenis Cloud Computing</i> .....	7
6.3. <i>Openstack</i> .....	9
6.4. <i>Sistem Administrator</i> .....	9
6.5. <i>Python</i> .....	10
6.6. <i>RESTful API</i> .....	10
6.7. <i>Backup</i> .....	11
6.8. <i>Ansible</i> .....	12
6.9. <i>Algoritma Threshold Adaptif</i> .....	12
6.10. Teknik Pengujian Perangkat Lunak .....	14
6.10.1. <i>BlackBox Testing</i> .....	14

6.10.2. <i>Performance Testing</i> .....	15
7. Metode Penelitian .....	16
7.1. Analisis Kebutuhan .....	16
7.2. Kerangka Kerja Penelitian.....	16
7.3. Perancangan Sistem.....	18
7.3.1. Desain Arsitektur Kerja Sistem.....	18
7.4. <i>Flowchart</i> Sistem .....	20
7.4.1. <i>Flowchart</i> Konfigurasi Otomatis .....	21
7.4.2. <i>Flowchart Front-end Web</i> .....	22
7.4.3. <i>Flowchart Back-end</i> .....	24
7.5. Rancangan ERD .....	26
7.6. Tampilan Sistem.....	26
7.6.1. <i>Menu User</i> .....	26
7.6.2. <i>Menu Sistem administrator</i> .....	30
7.7. Evaluasi Perancangan Sistem .....	34
8. Jadwal Pelaksanaan Penelitian.....	34
DAFTAR PUSTAKA .....	35

## DAFTAR GAMBAR

Gambar 6.10.2.1 Kerangka Kerja Sistem.....	17
Gambar 7.3.1.1 Desain Kerja Sistem.....	18
Gambar 7.4.1.1 Flowchart Konfigurasi Otomatis.....	21
Gambar 7.4.2.1 Flowchart Front-end Web .....	22
Gambar 7.4.3.1 <i>Flowchart Back-End</i> .....	24
Gambar 7.4.3.1 <i>Entity Relationship Diagram</i> Sistem.....	26
Gambar 7.6.1.1 Tampilan <i>form Sign up</i> .....	27
Gambar 7.6.1.2 Tampilan <i>form Sign in</i> .....	27
Gambar 7.6.1.3 Tampilan <i>dashboard</i> .....	28
Gambar 7.6.1.4 Tampilan <i>form Create Project</i> .....	28
Gambar 7.6.1.5 Tampilan setelah membuat project .....	29
Gambar 7.6.1.6 Tampilan detail <i>project</i> .....	29
Gambar 7.6.1.7 Tampilan <i>Backup history</i> .....	30
Gambar 7.6.2.1 Tampilan <i>form login administrator</i> .....	31
Gambar 7.6.2.2 Tampilan <i>dashboard administrator</i> .....	31
Gambar 7.6.2.3 Tampilan <i>list virtual mesin user</i> .....	32
Gambar 7.6.2.4 Tampilan <i>list aplikasi tersedia</i> .....	33
Gambar 7.6.2.5 Tampilan <i>menu monitoring</i> .....	33

## **DAFTAR TABEL**

Tabel 6.1 Tinjauan Studi .....	3
Tabel 6.2 Tabel Pengujian Black Box.....	15
Tabel 8.1 Rancangan Jadwal Pelaksanaan Penelitian .....	34

## 1. Latar Belakang

*Cloud computing* merupakan teknologi yang saat ini mulai berkembang dalam banyak aktivitas teknologi informasi. *Cloud computing* merupakan model komputasi yang semua sumber daya yang ada dalam layanan *cloud* dijalankan dengan media jaringan internet. Dengan adanya *cloud computing* memudahkan para pengguna dalam melakukan komputasi tanpa harus melakukan instalasi aplikasi pada komputer, pengguna hanya perlu mengaksesnya melalui internet. *Cloud computing* memiliki beberapa fasilitas yang dapat dipilih oleh pengguna sesuai kebutuhan pengguna seperti *Infrastructure as a Service* (IaaS) sebagai penyedia infrastruktur pada pengguna, *Platform as a Service* yang dapat digunakan oleh pengembang aplikasi untuk mengembangkan aplikasi yang akan dibuat tanpa perlu menyediakan infrastruktur, *database*, *framework* aplikasi dan lain sebagainya. Serta *Software as a Service* (PaaS) yang memberikan software yang siap digunakan oleh pengguna.

Dalam layanan *cloud* dikelola oleh seorang sistem *administrator* atau *sysadmin* yang bertugas menginstalasi dan menkonfigurasi *server*, *install* dan mengkonfigurasi *software* aplikasi, membuat dan mengelola *user*, *backup* dan *restore file*, konfigurasi keamanan *server*, serta *memonitor* keamanan jaringan agar layanan yang dibutuhkan dapat berjalan dengan baik. Permasalahan yang muncul adalah untuk melakukan hal tersebut sistem administrator harus mengkonfirmasi kebutuhan pengguna, serta untuk melakukan *instalasi* dan konfigurasi pada *virtual server* membutuhkan waktu yang cukup lama. Melakukan *backup* pada banyak pengguna *server virtual* dapat menyita banyak waktu dari seorang sistem administrator karena harus mengelola banyak *server virtual* pengguna. Performa kinerja *server* utama *cloud* menjadi sebuah hal yang penting pula, bila *server* utama penyimpanan data pengguna mengalami masalah maka *virtual* mesin yang ada pada *server* utama mengalami masalah pula. Sehingga sangat penting menjaga performa dan kondisi *server cloud*.

Dari masalah tersebut penelitian ini diharapkan dapat mempermudah tugas – tugas dari sistem administrator dengan sistem manajemen berbasis *cloud* yang dibangun menggunakan API dari *openstack* dan *flask python*, sistem manajemen



layanan web berbasis *PaaS* dapat mendukung kinerja seorang sistem *administrator* dalam mengelola setiap *virtual server* yang dimiliki. Sistem dapat melakukan instalasi dan konfigurasi secara otomatis bila ada pengguna baru yang membutuhkan sebuah server *virtual* untuk *website*, melakukan *backup* sistem pengguna secara berkala, untuk menjaga *data* para pengguna bila terjadi sesuatu yang tidak diinginkan dan sistem menerapkan metode *threshold* adaptif untuk mengoptimalkan kinerja *resource* sehingga sistem mengetahui tingkat kinerja yang optimal dalam mengelola *virtual* mesin yang ada. *Data* hasil *monitoring* setiap *virtual server* diolah untuk mendapatkan batas ambang adaptif dengan metode *threshold* adaptif. Batas ambang adaptif menjadi tolak ukur performa batas optimal *server* dalam melayani *virtual* mesin. Penerapan metode *threshold* adaptif menggunakan dasar jurnal “*A Heuristic Adaptive Threshold Algorithm on IaaS Clouds*”, sebagai acuan dalam melakukan penelitian ini. Metode *threshold* adaptif dapat memperbaiki konsumsi energi sebesar 10-20 persen. Algoritma *threshold* adaptif lebih efisien dan mampu beradaptasi dengan baik terhadap beban kerja dengan variabel ( Xia, Lan, & Xiao, 2015).

## **2. Rumusan Masalah**

Berdasarkan latar belakang, dapat dirumuskan beberapa permasalahan dalam penelitian ini yaitu:

1. Bagaimana sistem yang dibangun dapat mendukung pekerjaan sistem administrator dalam mengelola *server cloud*.
2. Bagaimana membangun sistem yang memfasilitasi *developer* dalam mengembangkan sistem yang dimiliki berbasis *cloud*.
3. Bagaimana *server cloud* dapat mengoptimasi performa sumber dayanya dengan metode *threshold adaptif*.

## **3. Tujuan Penelitian**

Penelitian ini memiliki beberapa tujuan yang ingin dicapai, berdasarkan rumusan masalah yang ditemukan, yaitu:

1. Membuat *developer* mengurangi beban *resource* dari komputer dalam mengembangkan aplikasi yang dibuat.
2. Mendukung kinerja sistem administrator dalam mengelola *server* agar kinerja sistem administrator menjadi optimal.
3. Sistem administrator dapat memantau performa sumber daya yang dimiliki pada *server cloud*.

#### 4. Batasan Masalah

Penelitian ini memiliki beberapa batasan masalah, guna fokus untuk mencapai tujuan yang sudah dijabarkan sebelumnya, yaitu:

1. Layanan *cloud* menggunakan *openstack* sebagai penyedia layanan *cloud computing*.
2. Sistem melayani *virtual server* untuk membuat layanan aplikasi *web*.
3. Sistem dibangun menggunakan *python* dengan *framework flask*.
4. Sistem tidak menyediakan *domain* untuk layanan pengguna.

#### 5. Manfaat Penelitian

Sistem yang dibuat dapat mendukung pekerjaan seorang *administrator* untuk mengelola *server* utama, serta memberikan wadah bagi para *developer* untuk bisa mengembangkan sistemnya secara *online* tanpa perlu menyiapkan *server*. Sehingga *developer* hanya fokus pada pengembangan aplikasinya.

#### 6. Tinjauan Pustaka

##### 6.1. Tinjauan Studi

Pada penelitian ini, peneliti menggunakan beberapa penelitian yang pernah dilakukan mengenai sistem layanan *PaaS* dan metode *adaptive threshold*. Beberapa penelitian tersebut yaitu:

Tabel 6.1 Tinjauan Studi

No	Judul	Tahun	Penulis
----	-------	-------	---------

1.	<i>A Dynamic Self-adaptive Resource-Load Evaluation Method in Cloud Computing</i>	2015	Liyun Zuo, Lei Shu, Shoubin Dong, Zhangbing Zhou, Lei Wang
2.	<i>Adaptive Resource Management for Service Workflows in Cloud Environments</i>	2013	Yi Wei, M. Brian Blake dan Iman Saleh
3.	<i>A Heuristic Adaptive Threshold Algorithm on IaaS Clouds</i>	2015	Qingxin Xia dan Yuqing Lan, Limin Xiao
4.	<i>The Design and Implementation of Resource Monitoring for Cloud Computing Service Platform</i>	2013	Lei Xiaojiang, Shang Yanlei
5.	<i>An Adaptive Threshold Method to Address Routing Issues in Delay-Tolerant Networks</i>	2011	Nicole Ng, Hwa Chang, Zhongjian Zou, Sai Tang

Dalam jurnal *A Dynamic Self-adaptive Resource-Load Evaluation Method in Cloud Computing* memaparkan Sumber daya *cloud* membuat indikator evaluasi statis tradisional tidak dapat menggambarkan keadaan sumber daya secara akurat, sehingga pada penelitian ini menggunakan tiga indikator dinamis yaitu jumlah permintaan sumber daya, kapasitas komputasi sumber daya dan kekuatan beban sumber daya untuk mengevaluasi keadaan beban sumber daya. Algoritma *self-adaptive* digunakan untuk mengevaluasi beban sumber daya. Dengan algoritma tersebut dapat membagi status *load resource* menjadi *overload*, normal dan *idle*. Pada sumber daya yang mengalami *overload* akan dimigrasi pada *load balance*, dan *idle resource* akan dilepas untuk menghemat energi bila waktu *idle* melebihi batas *threshold*. Sehingga pada penelitian ini mengusulkan dua *threshold* adaptif yang dipengaruhi oleh indikator evaluasi secara dinamis. Pada percobaan menampilkan *SDWM* dengan keuntungan yang sangat baik

dari waktu respon dan pemanfaatan sumber daya ketika sumber daya secara otomatis masuk dan keluar (Zuo, dkk, 2015).

Dalam jurnal *Adaptive Resource Management for Service Workflows in Cloud Environments* menjelaskan Manajemen sumber daya merupakan bagian yang sangat penting pada alur kerja manajemen aktivitas di dalam cloud. Dengan algoritma *adaptif Resources Management* untuk layanan alokasi dan dealokasi secara dinamis pada alur kerja sumberdaya sebelum mereka mengeksekusi pada sistem *cloud*. Tujuan dari jurnal ini, algoritma dapat menjaga jumlah total sumber daya yang dimiliki oleh masing-masing layanan pada tingkat yang diinginkan sehingga sumber daya tersebut tidak terlalu banyak melakukan komunikasi dan tidak memanfaatkan banyak sumber daya. Hasil prediksi adaptif digunakan untuk memandu algoritma dalam memilih kandidat sumber daya yang akan di lepas atau meminta sumber daya baru untuk dialokasikan. Dengan menjalankan simulasi pada *data* beban kerja secara sintetis. Algoritma tersebut melakukannya dengan baik daripada reaktif sumber daya *Management* yang ada dan dapat mencapai fluktuasi beban layanan yang lebih rendah dan kehilangan sumber daya (Wei, Blake, & Saleh, 2013).

Dalam jurnal *A Heuristic Adaptive Threshold Algorithm on IaaS Clouds* memaparkan teknologi untuk menghemat energi *IaaS* telah banyak menarik perhatian, namun bagi penyedia layanan cloud *IaaS* menjamin penghematan energi dan kinerja di bawah kondisi *Service Level Agreement (SLA)*. Pada jurnal ini menggunakan metode berbasis adaptif *threshold* untuk mengurangi *tradeoff* antara penghematan energi dan *SLA*, serta dapat menemukan ambang batas secara optimal. Penulis mengusulkan sebuah *framework* yang basisnya pada beban kerja, yang mengintegrasikan secara mulus metode prediksi berbasis pilihan dan model untuk mengukur hubungan antara biaya migrasi mesin *virtual (VM)* dan listrik saat mesin fisik mati. Algoritma *threshold* dirancang untuk menangkap ambang batas secara efektif. Metode ini dapat memperbaiki konsumsi energi sebesar 10-20 persen. Algoritma *threshold* adaptif lebih efisien dan mampu beradaptasi dengan baik terhadap beban kerja dengan variabel tinggi ( Xia, Lan, & Xiao, 2015).

Dalam jurnal “*The Design and Implementation of Resource Monitoring for Cloud Computing Service Platform*”. Membahas dengan *cloud computing* pengguna dapat menemukan solusi baru yang efektif untuk membangun sistem dan aplikasi berdasarkan kebutuhan mereka. karena *cloud computing* menghasilkan ketersediaan dan skalabilitas yang tinggi. pada bagian *background* tim tersebut mengembangkan *subsistem* yang digunakan untuk memonitor sumber daya, dengan bantuan dari *platform* yang dibuat, pada *layer* atas berjalan dengan lancar (Xiaojiang & Yanlei, 2013).

Pada jurnal *An Adaptive Threshold Method to Address Routing Issues in Delay-Tolerant Networks* memaparkan algoritma *routing* berbasis epidermi memberikan solusi dimana menggunakan beberapa salinan paket untuk meningkatkan probabilitas pengiriman paket sukses, namun banyak salinan paket tidak hanya meningkatkan beban jaringan tetapi juga menurunkan *throughput*, dari hal tersebut pada jurnal ini menggunakan metode *threshold*, metode *threshold* menggunakan pendekatan nilai statis. pada jurnal metode *threshold* adaptif mempengaruhi nilai *threshold* secara terus menerus sehingga dapat menyesuaikan dengan topologi secara dinamis. pada hasil simulasi tersebut memverifikasi bahwa nilai *threshold* adaptif melebihi metode *threshold* dengan mengurangi tingkat pengiriman paket. Pada penelitian ini menguji kinerja metode *threshold* adaptif melalui berbagai topologi sederhana, dengan metode *threshold* secara konsisten menunjukkan *throughput* yang tinggi dengan mengurangi tingkat pengiriman sekitar 3,9 detik. Oleh karena itu metode adaptif *threshold* merupakan pendekatan secara praktis untuk menangani masalah *routing* pada *Delay-Tolerant Networks(DTN)* (Ng, Chang, Zou, & Tang, 2011).

Dari kelima rangkuman jurnal yang menjadi acuan dalam melakukan penelitian ini, penelitian dilakukan lebih berfokus dalam penerapan *cloud computing* dalam mendukung kinerja sistem *administrator*, mempermudah para *developer* dalam mengembangkan aplikasi. Untuk penerapan metode *threshold* adaptif mengacu pada jurnal *A Heuristic Adaptive Threshold Algorithm on IaaS Cloud*, dimana metode *threshold* adaptif digunakan untuk menentukan ambang batas adaptif setiap *virtual*

*server*, bila *virtual server* terdeteksi memiliki *load* yang tinggi, *server* akan melakukan migrasi, sedangkan bila *server* mendeteksi *virtual* mesin keadaan diam dan tidak melakukan aktivitas, *server* akan menghapus *virtual* mesin. Dalam penelitian ini lebih mengarah menerapkan metode *threshold* adaptif untuk melakukan optimasi performa *main server cloud*, sehingga *server* dapat menentukan batas optimal dalam menangani jumlah pengguna yang dapat dilayani.

## **6.2. Cloud Computing**

*Cloud computing* mengacu pada aplikasi dan *service* yang berjalan dalam jaringan data terdistribusi dengan menggunakan sumber daya *virtual* dan *internet* akses protokol pada umumnya. Hal ini dibedakan pada gagasan sumber daya *virtual* dan detail dari mesin fisik sistem dalam *software* yang berjalan secara abstraksi dari *user*. (Sosinsky, 2011). *Cloud Computing* secara sederhana adalah layanan teknologi informasi yang bisa dimanfaatkan atau diakses oleh pelanggannya melalui jaringan *internet*. Komputasi awan adalah suatu konsep umum yang mencakup *SaaS*, *Web 2.0*, dan tren teknologi terbaru lain yang dikenal luas, dengan tema umum berupa ketergantungan terhadap Internet untuk memberikan kebutuhan komputasi pengguna. Sebagai contoh, *Google Apps* menyediakan aplikasi bisnis umum secara *sharing* yang diakses melalui suatu penjelajah web dengan perangkat lunak dan data yang tersimpan di *server*.

### **6.2.1. Jenis Cloud Computing**

#### **1. IaaS (Infrastructure as a Service)**

Terletak satu level lebih rendah dibanding *PaaS*. Ini adalah sebuah layanan yang “menyewakan” sumber daya teknologi informasi dasar, yang meliputi media penyimpanan, *processing power*, *memory*, sistem operasi, kapasitas jaringan dan lain – lain, yang dapat digunakan oleh penyewa untuk menjalankan aplikasi yang dimilikinya. Model bisnisnya mirip dengan penyedia *data center* yang menyewakan ruangan untuk *co-location*, tapi ini lebih ke *level* mikronya. Penyewa tidak perlu tahu, dengan mesin apa dan bagaimana caranya penyedia layanan menyediakan layanan

*IaaS*, yang penting permintaan mereka atas sumber daya dasar teknologi informasi itu dapat dipenuhi.

## 2. *Platform as a Service (PaaS)*

Konsepnya hampir serupa dengan *IaaS*. Namun *Platform* disini adalah penggunaan *operating system* dan infrastruktur pendukungnya. Yang cukup terkenal adalah layanan dari situs *Force.Com* serta layanan dari para vendor *server*. Seperti namanya, *PaaS* adalah layanan yang menyediakan modul – modul siap pakai yang dapat digunakan untuk mengembangkan sebuah aplikasi, yang tentu saja hanya bisa berjalan diatas *platform* tersebut. Seperti juga layanan *SaaS*, pengguna *PaaS* tidak memiliki kendali terhadap sumber daya komputasi dasar seperti *memory*, media penyimpanan, *processing power* dan lain-lain, yang semuanya diatur oleh *provider* layanan ini. Pionir di area ini adalah *Google AppEngine*, yang menyediakan berbagai *tools* untuk mengembangkan aplikasi di atas *platform Google*, dengan menggunakan bahasa pemrograman *Python* dan *Django*. Kemudian *Salesforce* juga menyediakan layanan *PaaS* melalui *Force.com*, menyediakan modul – modul untuk mengembangkan aplikasi diatas *platform Salesforce* yang menggunakan bahasa *Apex*. Dan mungkin yang jarang sekali kita ketahui, bahwa *Facebook* juga bisa dianggap menyediakan layanan *PaaS*, yang memungkinkan kita untuk membuat aplikasi diatasnya.

## 3. *Software as a Service (SaaS)*

Berada satu tingkat diatas *PaaS* dan *IaaS*, dimana disini yang ditawarkan adalah *software* atau suatu aplikasi bisnis tertentu. Contoh yang paling mutakhir adalah *SalesForce.Com*, *Service-Now.Com*, *Google Apps*, dsb. *SaaS* ini merupakan layanan *Cloud Computing* yang paling dahulu populer. *Software as a Service* ini merupakan evolusi lebih lanjut dari konsep *ASP (Application ServiceProvider)*. Sesuai namanya, *SaaS* memberikan kemudahan bagi pengguna untuk bisa memanfaatkan sumber daya perangkat lunak dengan cara berlangganan. Sehingga tidak perlu mengeluarkan investasi baik untuk *in house development* ataupun pembelian lisensi. Dengan cara berlangganan via *web*, pengguna dapat langsung menggunakan berbagai fitur yang

disediakan oleh penyedia layanan. Hanya saja dengan konsep *SaaS* ini, pelanggan tidak memiliki kendali penuh atas aplikasi yang mereka sewa. Hanya fitur-fitur aplikasi yang telah disediakan oleh penyedia saja yang dapat disewa oleh pelanggan.

### **6.3. Openstack**

*OpenStack* adalah sistem aplikasi *cloud* yang mengelola sumber daya seperti komputasi, penyimpanan dan jaringan, yang tersedia pada infrastruktur fisik seperti dalam sebuah fasilitas pusat – data (*data center*). *Admin* atau pengguna dapat mengendalikan dan melakukan *provisioning* atas sumber – daya ini melalui *dashboard* / antar-muka *web*. *Developer* dapat mengakses sumber daya tersebut melalui sejumlah *API* standar (Mulyana, 2017).

*Openstack* merupakan *platform cloud-computing open source* yang memungkinkan pengguna untuk membangun sebuah "IAAS" Infrastruktur sebagai *service cloud* yang bergerak secara massal pada komoditas *hardware* dan skala. *Openstack* mengontrol kolam besar komponen komputasi awan di seluruh *data center*, semua dikelola melalui *dashboard* yang menyediakan *administrator* kontrol penuh sambil memberikan pengguna kemampuan untuk sumber penyediaan melalui antarmuka *web*.

### **6.4. Sistem Administrator**

Sistem *administrator* atau *sysadmin* merupakan sebuah pekerjaan yang bertugas mengatur dan memelihara dan mengoperasikan sistem komputer dan jaringan komputer. Dalam perusahaan maupun sebuah organisasi sistem *administrator* sangat dibutuhkan dalam mengelola dan mengamankan *data* pada *server* yang dimiliki perusahaan (Collings & Kurt , 2015). Adapun tugas *sysadmin* sebagai berikut:

1. *Menginstalasi* dan menkonfigurasi *server*.
2. *Menginstall* dan mengkonfigurasi *software* aplikasi.
3. Membuat dan mengelola *user*.
4. *Backup* dan *restore file*.
5. Konfigurasi keamanan *server*.



## 6. Memonitor keamanan jaringan

### 6.5. *Python*

*Python* merupakan bahasa pemrograman yang sangat *powerful* yang memiliki beberapa kesamaan dengan *Fortran*, salah satu bahasa pemrograman paling awal, namun lebih baik dari *fortran*. *Python* memungkinkan anda untuk menggunakan variabel tanpa perlu mendeklarasikannya. Sehingga kita tidak dipaksa untuk mendefinisikan *class* dengan *python*.

*Python* dikembangkan oleh Guido van Rossum pada tahun 1990 di CWI, Amsterdam sebagai kelanjutan dari bahasa pemrograman ABC. Versi terakhir yang dikeluarkan CWI adalah 1.2. (Doty, 2008).

Tahun 1995, Guido pindah ke CNRI sambil terus melanjutkan pengembangan *Python*. Versi terakhir yang dikeluarkan adalah 1.6. Tahun 2000, Guido dan para pengembang inti *Python* pindah ke *BeOpen.com* yang merupakan sebuah perusahaan komersial dan membentuk *BeOpen PythonLabs*. *Python 2.0* dikeluarkan oleh *BeOpen*. Setelah mengeluarkan *Python 2.0*, Guido dan beberapa anggota tim *PythonLabs* pindah ke *DigitalCreations*.

### 6.6. *RESTful API*

*REST* (*REpresentational State Transfer*) merupakan standar arsitektur komunikasi berbasis *web* yang sering diterapkan dalam pengembangan layanan berbasis *web*. Umumnya menggunakan *HTTP* (*Hypertext Transfer Protocol*) sebagai *protocol* untuk komunikasi *data*. *REST* pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000.

*REST API* merupakan *web service* yang bertujuan untuk mendukung kebutuhan *server web* pada suatu kebutuhan situs atau aplikasi lainnya. program *client* menggunakan *Application Programming Interface* (*API*) untuk berkomunikasi dengan layanan *web*. Secara umum, *API* mengekspos seperangkat data dan fungsi untuk

memfasilitasi interaksi antara program komputer dan memungkinkan mereka saling bertukar informasi (Masse, 2012).

Dalam pengaplikasiannya, *REST* lebih banyak digunakan untuk *web service* yang berorientasi pada *resource*. Maksud orientasi pada *resource* adalah orientasi yang menyediakan *resource – resource* sebagai layanannya dan bukan kumpulan – kumpulan dari aktifitas yang mengolah *resource* tersebut.

### **6.7. Backup**

*Backup data* merupakan proses memindahkan *data* dari *primary* komputer ke penyimpanan yang terpisah. Jika *data* yang asli tersebut hilang maupun rusak, kita dapat *merestore* kembali informasi dari penyimpanan. *File* yang paling penting untuk dilakukan *backup* adalah *file data*, secara berkala kita harus mencadangkan seluruh sistem jika terjadi bencana besar. (Corporation, Exabyte, 2004). Pencadangan berkala ini harus mencakup *file* sistem yang berisi informasi pengguna. Ada beberapa tipe dalam melakukan backup antara lain :

#### *1. Full Backups*

Menyalin semua *file* sistem yang ada pada *file* sistem, *file* perangkat lunak, dan *file data*. Kita dapat melakukan *backup* secara mingguan atau bulanan, dengan cara *full backups* dan data yang ada dapat dipulihkan keseluruhan sistem ketika terjadi bencana.

#### *2. Partial Backups*

Menyalin semua *file* yang telah ditambahkan atau diubah sejak pencadangan terakhir. Ada 2 tipe utama pada *partial backup* yaitu :

- a. *Incremental* yaitu *file* ditambah atau diubah sejak *backup* terakhir atau sebagian terakhir.
- b. *Differential* yaitu *file* ditambah atau diubah sejak *full backup* terakhir.

## 6.8. *Ansible*

*Ansible* merupakan sebuah software yang bisa membantu seorang sistem administrator untuk melakukan otomasi pada server. *ansible* merupakan teknologi yang digunakan untuk melakukan otomasi, memudahkan dalam melakukan konfigurasi server, tujuan dibuat *ansible* membuat hal tersebut menjadi sederhana dan mudah, namun tetap fokus pada keamanan dan keandalan dalam melakukan otomasi. *ansible* menggunakan *OpenSSH* untuk transportasi ( dengan mode *socket* yang cepat). (RedHat.Inc, 2017)

Dengan *ansible* *sysadmin* dapat melakukan instalasi, *deployment* hingga melakukan *update server*. Sistem kerja yang dimiliki oleh *ansible* membutuhkan koneksi khusus berupa *SSH*. *Ansible* bekerja di koneksi *SSH remote client* yang ingin di *deploy* atau dilakukan otomasi. Pada *ansible* memerlukan *inventory* atau *data server* tujuan untuk dapat dilakukan otomasi. Pada penerapannya, *ansible* menggunakan *playbook* dan *roles*, dimana konfigurasi tersebut dalam format *markup YAML* dan *environment* variabel dapat ditulis dalam bentuk *JSON*.

*Ansible* dirancang untuk memudahkan para sistem administrator dan para pakar IT mengelola lingkungan server dengan mudah. *ansible* mengelola mesin dengan cara yang tidak biasa, tidak pernah bertanya cara melakukan *upgrade daemon* jarak jauh atau masalah karena tidak dapat mengelola sistem karena *daemon* sistem terhapus.

*Ansible* merupakan salah satu jenis *Configuration Management Tools* yang dapat digunakan merubah proses infrastruktur manajemen dari *program* manual menjadi otomatis. Dalam zaman *cloud* kehadiran *ansible* membantu para sistem administrator atau para *devops* dalam instalasi dan konfigurasi server dengan otomatis, oleh karena itu *ansible* menjadi satu platform yang digunakan untuk mengelola server – server.

## 6.9. Algoritma *Threshold Adaptif*

Untuk mendeteksi fluktuasi tidak normal terutama dalam membandingkan nilai prediksi pada titik sampling dan nilai beban kerja dengan menggunakan metode

*threshold* adaptif, menetapkan rentang fluktuasi yang normal untuk prediksi beban kerja. Bila *data* beban kerja dalam kisaran yang ditentukan maka dianggap sebagai fluktuasi normal, sebaliknya bila data beban berada di luar jangkauan maka akan dianggap sebagai data beban kerja tidak normal. Sebelum membahas mengenai metode *threshold* adaptif, kita harus memahami karakteristik beban kerja ( Xia, Lan, & Xiao, 2015).

Beban kerja aplikasi secara langsung mencerminkan konsumsi sumber daya fisik dari *server virtual*. Menganalisa karakteristik beban kerja dapat menggunakan sejumlah *data* riil, sebelum melakukan menguji dengan *threshold* adaptif, kita akan mendefinisikan model beban kerja.

1. Mendefinisikan beban kerja *virtual* mesin, di mana  $V$  merupakan keseluruhan dari *virtual* mesin,  $v_i$  merupakan beban kerja spesifik dari virtual mesin.

$$V = \{v_1, v_2, v_i\}$$

2. Beban kerja *virtual* mesin secara spesifik  $v_i$  :

$$v_i = \{ \alpha_i, \beta_i, \gamma_i \}$$

3. Rata – rata beban kerja didefinisikan dengan parameter  $\varepsilon$  panjang beban kerja ( $\alpha$ ) , *CPU* ( $\beta$ ) , penggunaan *memory* ( $\gamma$ ) dengan menggunakan kuantitas dari sampel beban kerja nyata.

$$\varepsilon\beta_i = \frac{\Sigma\beta_i}{\alpha_i}, \varepsilon\gamma_i = \frac{\Sigma\gamma_i}{\alpha_i}$$

4. Setelah mendapatkan beban kerja dari virtual mesin. Lalu menggunakan *SLATAH* ( *SLA Time per Active Host*) untuk mengukur tingkat pelanggaran *SLA* ( *Service Level Agreement*)

$$SLATAH = \frac{1}{N} \sum_{i=1}^N \frac{T_{s_i}}{T_{a_i}}$$

Di mana  $N$  merupakan nomor mesin fisik,  $T_{s_i}$  merupakan total waktu di mana mesin fisik telah mengalami penggunaan 100% yang mengarah pada pelanggaran *SLA*,  $T_{a_i}$  jumlah mesin fisik yang ada dalam keadaan aktif.

5. Penyesuaian ambang adaptif (*adaptive threshold*) untuk mendeteksi fluktuasi abnormal terutama untuk membandingkan nilai prediksi pada titik sampling dan nilai beban kerja riil, jika nilai absolut nilai D melebihi batas tertentu, maka tahap ini disebut dengan fluktuasi tidak normal. Menentukan batas memiliki dua cara yaitu dengan cara batas empiris dan batas adaptif, dalam penelitian ini peneliti menggunakan batas adaptif dengan menentukan :

$$D = (d_i) = \frac{T_i - S_i}{S_i}$$

Di mana  $T_i$  merupakan parameter Sumber daya virtual mesin dan  $S_i$  merupakan prediksi nilai dari beban kerja virtual mesin. Untuk dapat dikondisikan limit secara adaptif menggunakan

$$a = \frac{\sum_{i=0}^n (S_i - S'_i)}{n}$$

Di mana  $n$  merupakan nomor sampel dari nilai poin per hari,  $S_i$  merupakan nilai riil dari beban kerja sistem pada sampel waktu  $i$ ,  $S'_i$  merupakan nilai prediksi dari beban kerja sistem pada sampel waktu ke  $i$ .

## **6.10. Teknik Pengujian Perangkat Lunak**

Terdapat strategi pengujian menurut (Everett & Jr, 2007) yang dibagi menjadi 4 bagian utama yaitu *Static Testing*, *White Box Testing*, *Black Box Testing*, dan *Performance Testing*. Dalam penelitian ini menggunakan teknik pengujian *black box* dan *Performance Testing*.

### **6.10.1. BlackBox Testing**

*Black Box Testing* atau dikenal sebagai “*Behaviour Testing*” merupakan suatu metode pengujian yang digunakan untuk menguji *executable code* dari suatu perangkat lunak terhadap perilakunya. Pendekatan *Black Box Testing* dapat dilakukan jika kita sudah memiliki *executable code*. Orang-orang yang terlibat dalam *Black Box Testing* adalah *tester*, *end-user*, dan *developer*.

Fokus dari pengujian ini ialah pada kebutuhan fungsional perangkat lunak, sehingga memungkinkan tester mendapatkan serangkaian kondisi *input* yang

sepenuhnya menggunakan semua persyaratan fungsional untuk suatu untuk program. Kesalahan yang ditemukan dalam pengujian, nantinya dapat disimpulkan apakah kesalahan tersebut murni dikarenakan kesalahan dari aplikasi atau kesalahan implementasi dari tester.

Tabel 6.2 Tabel Pengujian Black Box

Identifikasi		
Nama Kasus Uji		
Deskripsi		
Kondisi Awal		
Tanggal Pengujian		
Penguji		
Skenario		
1.		
2.		
(Dst...)		
Hasil Yang Diharapkan	Hasil Yang Didapatkan	Kesimpulan

Untuk pengujian antarmuka pengguna atau rancangan skenario pengujian *black box* dari sistem ini, dilakukan dua jenis pengujian yaitu pengujian secara *happy path* yaitu pengujian yang dilakukan dengan cara yang benar, serta pengujian secara *alternative path* yaitu mencoba segala kemungkinan yang mungkin terjadi pada sistem.

#### **6.10.2. Performance Testing**

Teknik pengujian memvalidasi perilaku perangkat lunak terhadap teknik pengujian software dari sisi kecepatan. Kecepatan ini dalam konteks pengujian dalam mengukur waktu respon perangkat lunak ketika berada jumlah kerja yang berlebih yang biasa dikenal dengan beban kerja. Untuk memperlihatkan kecepatan sebenarnya sebuah perangkat lunak harus dilakukan pengujian *performance testing*.

Tujuan dari *performance testing* untuk memvalidasi kecepatan sebuah perangkat lunak terhadap kebutuhan sistem yang cepat. Secara umum harus mendefinisikan kombinasi waktu *respons* dan beban kerja

## **7. Metode Penelitian**

Bagian ini akan menjelaskan mengenai langkah – langkah yang akan dilakukan dalam merancang sistem manajemen layanan *web* berbasis *Platform as a Service (PaaS)* dengan *API openstack*.

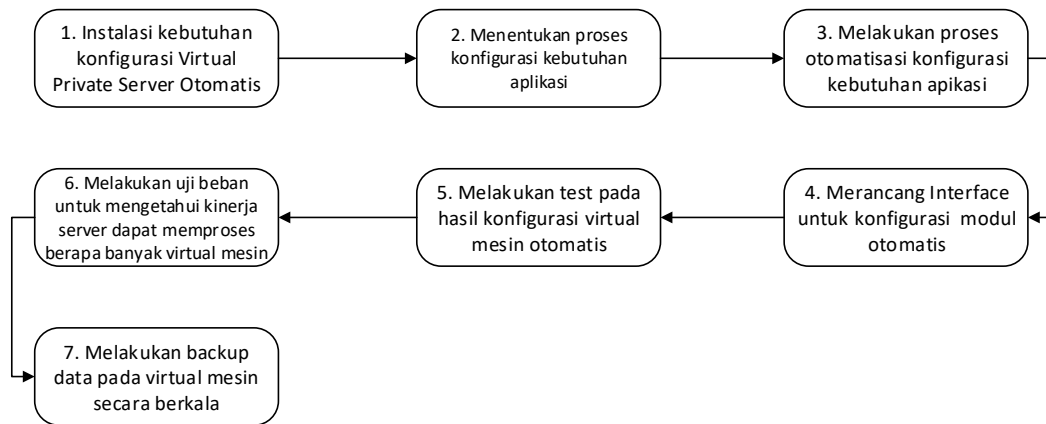
### **7.1. Analisis Kebutuhan**

Pada tahap ini dilakukan analisis kebutuhan sistem, analisis kebutuhan sistem meliputi data yang digunakan, pembelajaran dari referensi yang sudah ada dan perangkat yang digunakan baik perangkat lunak maupun perangkat keras:

1. Tahapan pertama yang dilakukan dalam penelitian ini adalah mengidentifikasi permasalahan. Tahap ini merupakan tahap yang paling penting dalam penelitian karena jalannya penelitian didasarkan atas permasalahan yang terjadi. Setelah menentukan masalah yang terjadi, tahapan yang diperlukan selanjutnya adalah menentukan rumusan masalah dan tujuan yang ingin dicapai dalam penelitian. Pada penelitian ini identifikasi permasalahan dilakukan dengan menggunakan teknik observasi, dari teknik ini maka akan dapat diketahui mengenai keluhan – keluhan yang ada di lapangan.
2. Tahap kedua yang dilakukan dalam metodologi penelitian ini adalah studi literatur. Studi literatur dilakukan dengan mengambil literatur – literatur pendukung dari jurnal – jurnal ilmiah, baik jurnal dalam negeri ataupun jurnal luar negeri dan dari beberapa buku. Dalam studi literatur ini, penulis mencari sumber terkait permasalahan – permasalahan yang perlu menjadi perbaikan dalam penelitian selanjutnya.

### **7.2. Kerangka Kerja Penelitian**

Bagian ini menjelaskan tentang bagaimana penelitian ini dilakukan. Berikut adalah beberapa proses penting yang dilakukan :



Gambar 6.10.2.1 Kerangka Kerja Sistem

#### 1. Instalasi kebutuhan konfigurasi VPS (*Virtual Private Server*)

Tahap pertama dalam penelitian ini dengan mempersiapkan sistem dan *software* yang dibutuhkan dalam mengkonfigurasi kebutuhan *web server* dalam menginstalasi modul-modul kebutuhan sistem seperti sistem operasi, *database*, *framework*, *web server*.

#### 2. Menentukan proses konfigurasi kebutuhan aplikasi

Pada tahap ini mempersiapkan kebutuhan konfigurasi untuk membangun sebuah *web server*, dari *web service* sampai *database* yang diperlukan untuk menampung setiap user.

#### 3. Melakukan proses otomatisasi konfigurasi modul aplikasi

Pada tahap ini melakukan otomatisasi dalam mengkonfigurasi sebuah *web server*. Dimana proses otomatisasi menggunakan *API* dari *Ansible*.

#### 4. Merancang *web front-end* untuk user

Pada tahap ini bila semua konfigurasi sudah berjalan dengan baik, dibuat *web* yang dapat digunakan oleh pengguna mengatur kebutuhan web yang dibuat serta dapat melihat informasi dari setiap *VPS* yang dimiliki.

#### 5. Melakukan *test* pada hasil konfigurasi



Pada tahap ini menguji hasil konfigurasi dari *web interface*. Untuk menguji konfigurasi berjalan dengan baik atau tidak

#### 6. Melakukan uji beban *server*

beban ini dilakukan untuk menguji tingkat kemampuan *server* dalam memberikan pelayanan pada pengembangan aplikasi. Bila *server* tidak sanggup untuk melayani pengembang baru akan menolak menyediakan layanan.

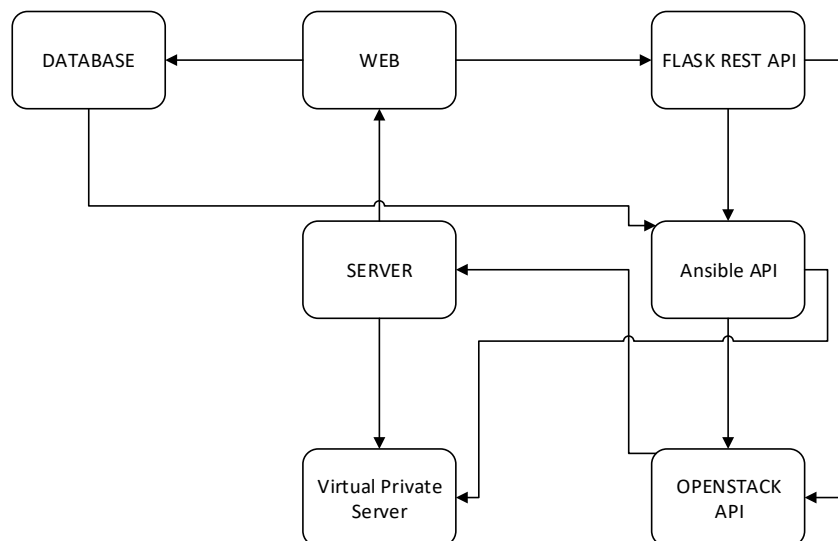
#### 7. Melakukan *Backup Server*

Ketika *server user* sudah berjalan, sistem akan melakukan *backup* data pengguna untuk menjaga ketersediaan *data* pengguna bila terjadi sesuatu yang dapat merugikan pengguna, seperti *file* data hilang atau rusak, dapat dikembalikan dengan *restore* pada sistem dengan menggunakan waktu *check point*.

### 7.3. Perancangan Sistem

Pada tahap Perancangan sistem, dilakukan berdasarkan hasil analisa kebutuhan sistem yang sudah dilakukan sebelumnya. Hal ini dilakukan agar, perancangan tidak keluar dari tujuan sistem yang dikembangkan.

#### 7.3.1. Desain Arsitektur Kerja Sistem



Gambar 7.3.1.1 Desain Kerja Sistem

Pada gambar 7.3.1.1 merupakan hubungan antara setiap perangkat lunak yang ada pada server utama. Berikut ini merupakan penjelasannya :

1. *Web dan database*

*Database* digunakan untuk menyimpan informasi pengguna serta informasi pada *VPS* yang dimiliki oleh setiap pengguna.

2. *Web dan framework flask REST API*

Flask REST API digunakan untuk menghubungkan web *front-end* dengan *back-end*. Dimana *back-end* dibuat dengan menggunakan *framework flask* dari *python*, selanjutnya dibuat *API* tersendiri agar web *front-end* dapat mengirim dan menerima informasi serta konfigurasi yang dilakukan yang nantinya akan diproses oleh *back-end server*.

3. *Database dan ansible API*

Hubungan *database* dengan *ansible API* akan mengambil informasi pengguna berupa *username*, *email* dan *password* yang nantinya akan dimasukkan pada konfigurasi dalam *virtual machine* yang selanjutnya sebagai *super admin* pada *VPS* yang dibuat.

4. *Framework flask REST API dan asible API*

Hubungan *framework flask REST API* dengan *asible API*. Ketika *user* melakukan konfigurasi pada *web* utama akan dikirim melalui *REST API flask* selanjutnya konfigurasi tersebut akan digunakan oleh *API ansible* untuk mengkonfigurasi *virtual machine* yang dibuat.

5. *Framework flask dan openstack API*

Hubungan *framework flask* dan *openstack API* adalah pada *flask* dikonfigurasi untuk dapat terhubung dengan *server devstack*, dimana *server openstack* yang mengelola dalam membuat sebuah *virtual machine*.

6. *Openstack API dan server*

*API openstack* digunakan untuk terhubung dengan server *back-end* yang dibuat dengan *framework flask*. Untuk dapat mengembangkan aplikasi *openstack* dari bahasa pemrograman yang berbeda.

#### 7. *Ansible API* dan *virtual server*

Hubungan *ansible API* dengan *virtual* mesin adalah *ansible* akan mengirimkan konfigurasi pada *VPS* melalui *SSH* yang yang dibuat. Selanjutnya pada *virtual* mesin akan melakukan konfigurasi yang diperlukan untuk membangun sebuah *web* aplikasi secara otomatis. Pengguna hanya perlu memilih keperluan yang ada *menu web front-end*.

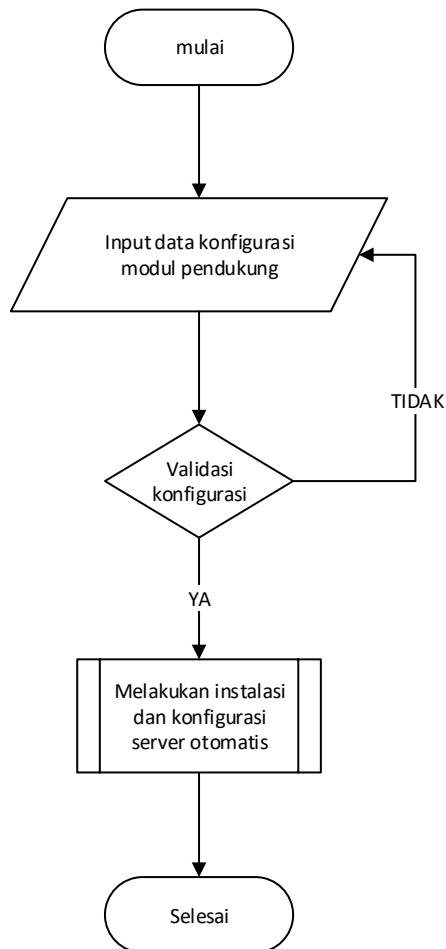
#### 8. *Server* dan *Virtual* Mesin

Hubungan antara *server* dan *virtual* mesin adalah *server* mengelola dan memberikan *resource* pada *virtual* mesin serta mengatur konektivitas setiap *virtual* mesin yang dimiliki.

### 7.4. **Flowchart Sistem**

Pada bagian ini akan menjelaskan proses yang dilakukan oleh perangkat lunak, bagaimana proses tersebut berjalan yang akan dijelaskan pada setiap *flowchart* berikut ini:

#### 7.4.1. Flowchart Konfigurasi Otomatis

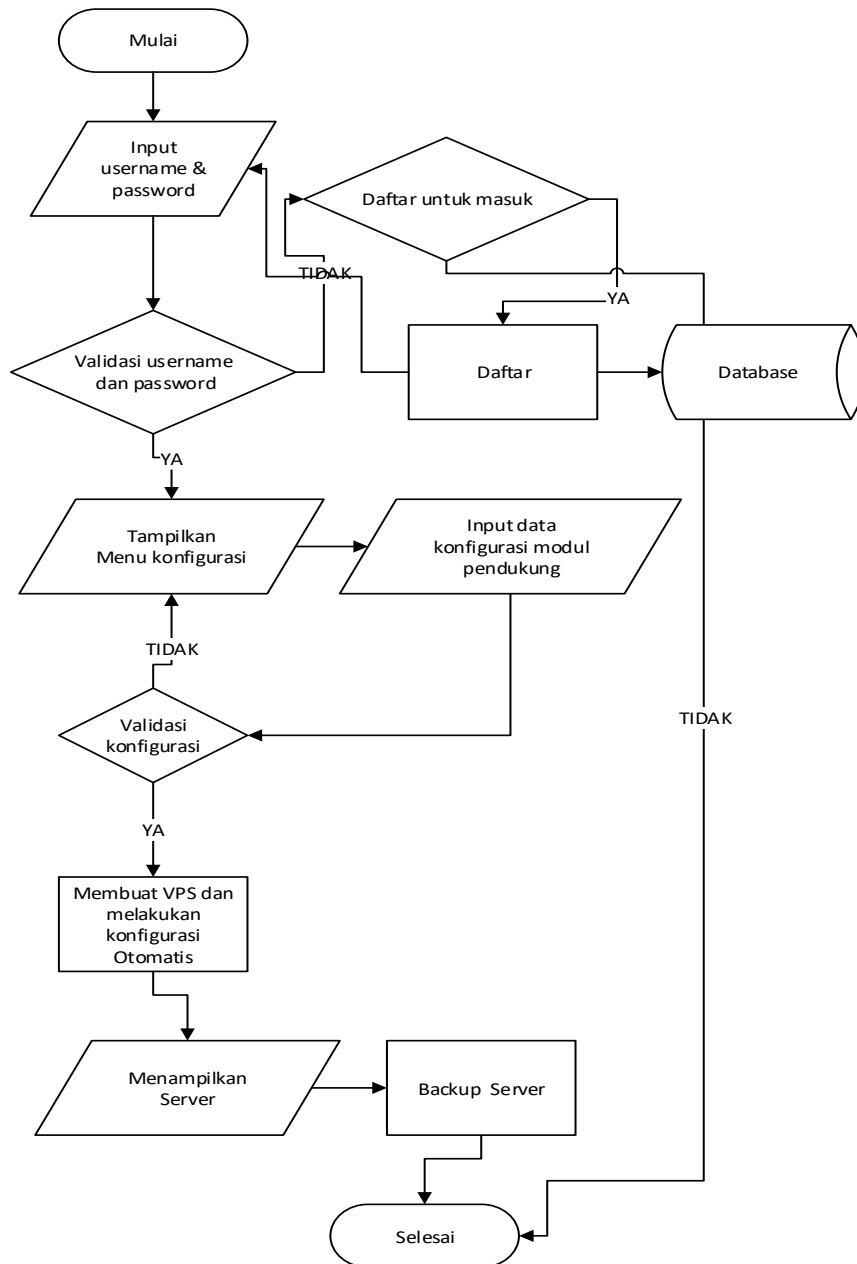


Gambar 7.4.1.1 Flowchart Konfigurasi Otomatis

Pada desain *flowchart* 7.4.1.1 merupakan garis besar gambaran *system* yang akan dikerjakan, bagaimana alur *platform* yang dibuat dapat mengotomatitasi dalam instalasi serta konfigurasi sebuah layanan pada *virtual machine*. Mulai dari memasukkan aplikasi pendukung yang ingin digunakan seperti *database*, *web service*, *username* dan *e-mail administrator* layanan aplikasi. Kemudian dari hasil *input* tersebut diolah oleh *flask API* yang dibuat untuk dimasukkan ke dalam konfigurasi yang terdapat ada *Ansible API*, selanjutnya dari *ansible API* dengan menggunakan *SSH* akan melakukan instalasi konfigurasi pada *virtual server* yang didapat setiap *user*. Bila

konfigurasi telah selesai *user* akan menerima sebuah *ip public* untuk dapat mengakses layanan aplikasi.

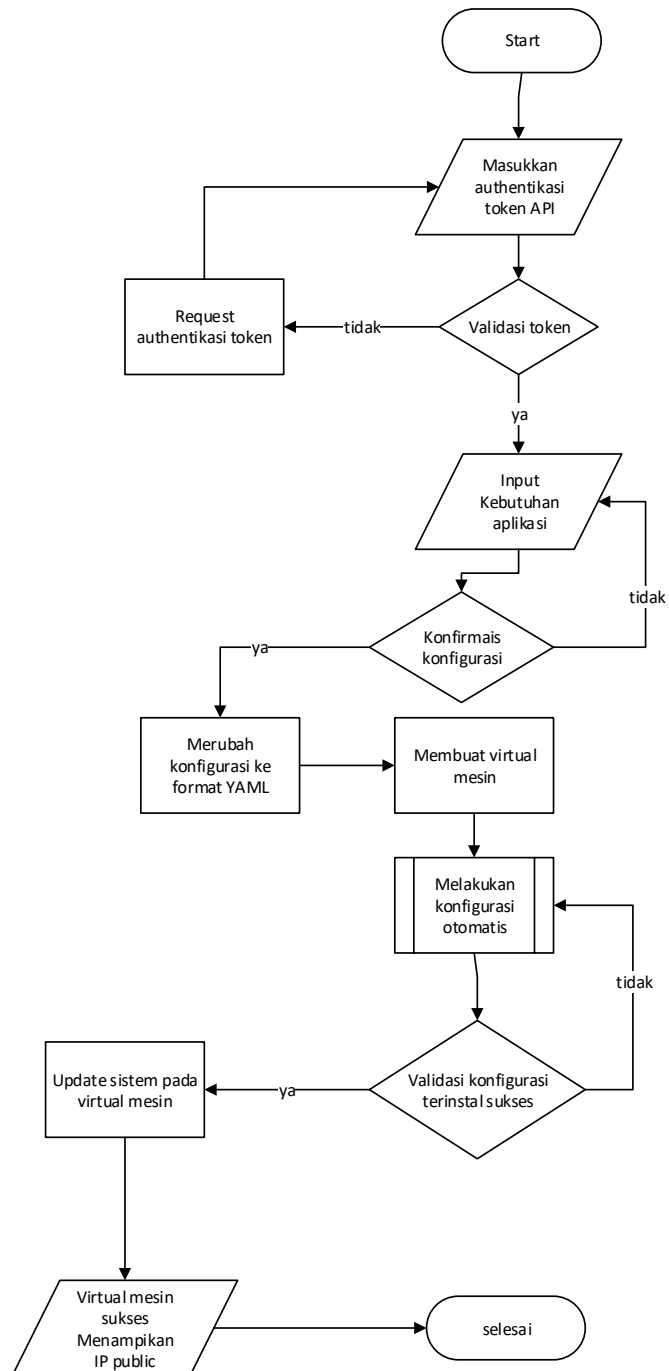
#### 7.4.2. Flowchart Front-end Web



Gambar 7.4.2.1 Flowchart Front-end Web

Pada desain *flowchart* diatas merupakan alur kerja *web* dari sisi *front-side* atau pada sisi *user*. Setelah *user* melakukan registrasi pada *website*, selanjutnya *user* akan diarahkan ke *menu* instalasi dan konfigurasi dalam membangun layanan *virtual* mesin. Menu akan dibuat secara *user friendly* sehingga *user* hanya perlu memilih *menu* konfigurasi *system*. Ketika *user* sudah selesai memilih menu konfigurasi, *system* akan memvalidasi informasi yang dimasukkan oleh *user*, bila sudah benar *system* akan mengirimkan hasil instalasi dan konfigurasi dalam sebuah *data* berbentuk *JSON* ke dalam *system* yang nantinya akan dikirim ke *virtual sever* untuk melakukan konfigurasi dan instalasi modul kebutuhan aplikasi selanjutnya sistem akan melakukan *backup* berkala untuk menjaga keamanan data pengguna bila terjadi satu kesalahan dapat dilakukan *restore data*.

### 7.4.3. Flowchart Back-end

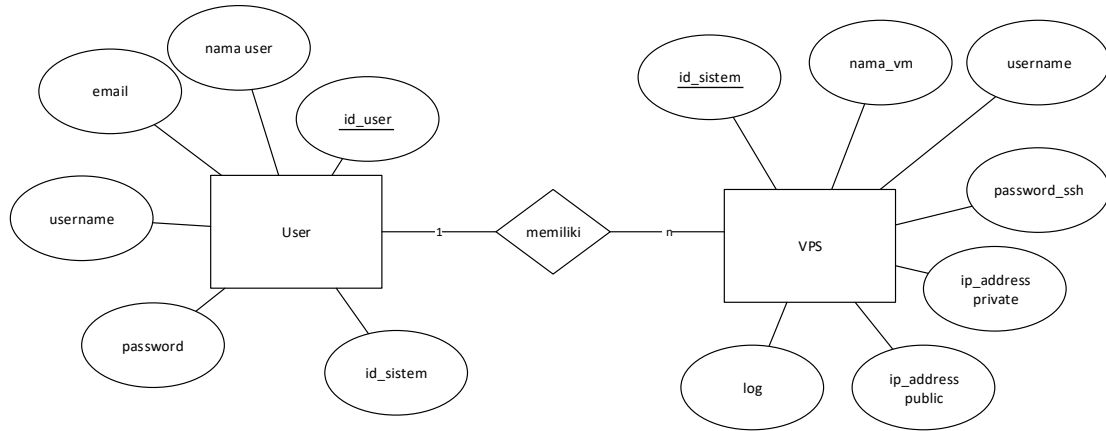


Gambar 7.4.3.1 Flowchart Back-End

Pada gambar flowchart 7.4..3.1 menjelaskan alur kerja sistem *back-end* sistem. Tahap pertama yang harus dilakukan adalah melakukan komunikasi dengan *openstack* agar dapat menggunakan *API* dari *openstack*. Melakukan komunikasi dengan melakukan *request token autentikasi*, hal ini dimaksudkan agar setiap komunikasi yang dibuat dapat melakukan *update data* pada *API* dan yang memiliki *token* dari *autentikasi* saja yang dapat berkomunikasi. Selanjutnya setelah melakukan *request*, sistem tetap akan melakukan validasi *token id* dari *openstack*, karena setiap melakukan *request token* memiliki batas waktu yang ditentukan. Setelah bisa melakukan komunikasi dengan *openstack* selanjutnya menerima masukan konfigurasi dari *front-end* dan sudah divalidasi oleh administrator, selanjutnya merubah setiap masukan konfigurasi dalam ke *YAML*, *YAML* merupakan format untuk memberi perintah pada *ansible* agar dapat melakukan instalasi maupun konfigurasi secara otomatis. Setelah *file* siap, sistem akan membuat *virtual* mesin dan mengirimkan konfigurasi tersebut ke *virtual* mesin dan langsung melakukan konfigurasi secara otomatis. Selanjutnya sistem akan kembali melakukan mengecek apakah sudah semua *terinstall* dan *terkonfigurasi* dengan baik, sehingga bila ada gagal dalam dilakukan *instalasi* kembali. Setelah semua dicek dan sukses maka selanjutnya sistem akan menampilkan *virtual* mesin siap dan sebuah *ip public* untuk diakses oleh *developer* dan *ssh key* bila *developer* memiliki kebutuhan aplikasi khusus yang tidak dimiliki oleh sistem.



## 7.5. Rancangan ERD



Gambar 7.4.3.1 *Entity Relationship Diagram* Sistem

Pada desain *ERD* diatas menjelaskan hubungan user dengan *VPS* yang dimiliki user memiliki sebuah vps dengan satu *ip public* untuk dapat mengakses *server*. Serta pada *vps* akan menyimpan *log* keadaan *vps* mulai dari kinerja dan *resource* yang dimiliki.

## 7.6. Tampilan Sistem

Pada tahap ini merupakan gambaran *menu* dan *tool – tool* dari sistem yang didapat oleh *user* dan administrator.

### 7.6.1. Menu User

Pada bagian ini menjelaskan *menu – menu* yang di dapat oleh pengguna pada sistem *PaaS* yang dibangun.

The screenshot shows a web form titled "IBRESWCloud" with the heading "Sign Up". It contains four input fields: "Nama Lengkap :", "Email :", "Username :", and "Password :". Below the fields are two buttons: "Sign Up" and "Back To Login".

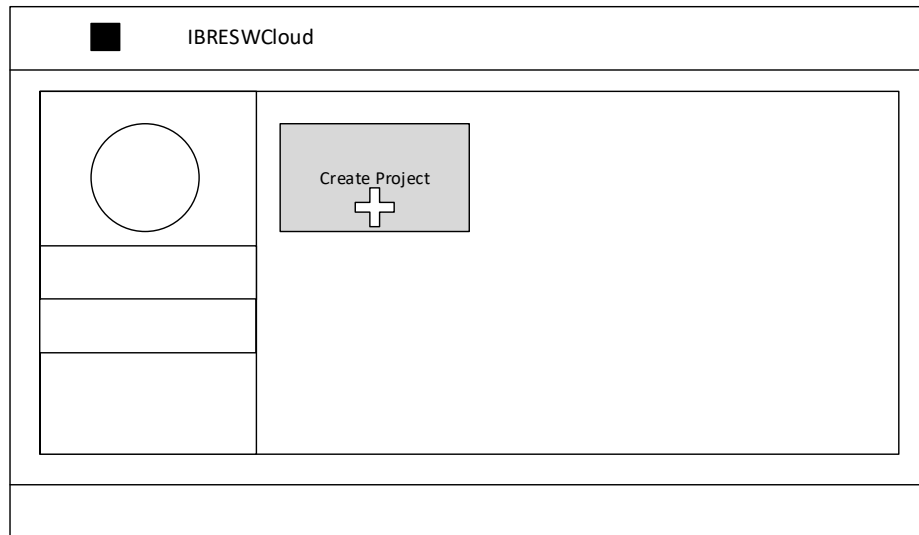
Gambar 7.6.1.1 Tampilan *form Sign up*

*Menu* pertama yaitu *form* untuk melakukan registrasi ke sistem, pada tahap ini pengguna melakukan registrasi dengan memasukkan *email*, *username*, dan *password*.

The screenshot shows a web form titled "IBRESWCloud" with the heading "Sign In". It contains two input fields: "Username :" and "Password :". Below the fields is a "Login" button. At the bottom, there are two links: "Register Now!" and "Forgot Password ?".

Gambar 7.6.1.2 Tampilan *form Sign in*

Bila pengguna sudah memiliki akun, selanjutnya akan melakukan *login* pada *menu sign in*.



Gambar 7.6.1.3 Tampilan *dashboard*

Pada gambar 7.6.1.3 menampilkan *dashboard user*, menu utama dari *dashboard* adalah membuat *project* baru, pada menu ini *user* dapat membuat serta melihat *project* yang sudah pernah dibuat oleh *user* tersebut.

Gambar 7.6.1.4 Tampilan *form Create Project*

Pada gambar 7.6.1.4 merupakan tampilan *form* untuk membuat *project* baru, setelah memilih *create project* selanjutnya akan tampil *form* seperti pada gambar

7.6.1.4. *User* selanjutnya *menginput* kebutuhan dari aplikasi *website* yang sedang dikerjakan, seperti sistem operasi, *database*, *web server* yang dibutuhkan dan lainnya.

The screenshot shows the IBRESWCloud web interface. At the top, there is a header bar with a black square icon and the text 'IBRESWCloud'. Below the header, the main content area is divided into two sections. On the left, there is a vertical sidebar containing a circular profile picture placeholder and three empty rectangular input fields. The main content area on the right contains a gray rectangular box labeled 'Project Name' and a button labeled 'Create Project' with a plus sign icon.

Gambar 7.6.1.5 Tampilan setelah membuat project

Pada gambar 7.6.1.5 menjelaskan ketika *user* telah berhasil membuat sebuah *project* akan tampil seperti gambar 7.6.1.5.

The screenshot shows the IBRESWCloud web interface in a detail view. The header bar at the top contains a black square icon and the text 'IBRESWCloud'. The main content area is divided into two sections. On the left, there is a vertical sidebar containing a circular profile picture placeholder and three empty rectangular input fields. The main content area on the right contains a large gray rectangular box labeled 'Detail' and a smaller gray rectangular box labeled 'Project Name'.

Gambar 7.6.1.6 Tampilan detail *project*

Pada gambar 7.6.1.6 menjelaskan ketika *user* memilih *project* yang telah dibuat, sistem akan menampilkan detail dari *project* tersebut, seperti status *virtual* mesin, *ip public* yang dapat di akses, *backup* terakhir dan lainnya.

The screenshot shows the IBRESWCloud web interface. At the top, there is a header with the logo and the text 'IBRESWCloud'. Below this, the main content area is titled 'BACKUP HISTORY'. To the right of the title is a search bar with the placeholder text 'Search : tgl'. Below the search bar is a table with the following columns: 'No', 'tanggal', 'Log', and 'Aksi'. The 'Log' column contains a modal dialog box titled 'Konfirmasi restore' with the text 'Pada tanggal : xx-xx-xxxx' and two buttons: 'Tidak' and 'Ya'. The 'Aksi' column contains two buttons: 'restore' and 'Delete'.

Gambar 7.6.1.7 Tampilan *Backup history*

Pada gambar 7.6.1.7 merupakan *form* yang menampilkan *backup history*. Sehingga bila terjadi kesalahan saat mengembangkan aplikasi, *user* dapat kembali *merestore* pada hasil *backup* sebelumnya.

## 7.6.2. Menu Sistem *administrator*

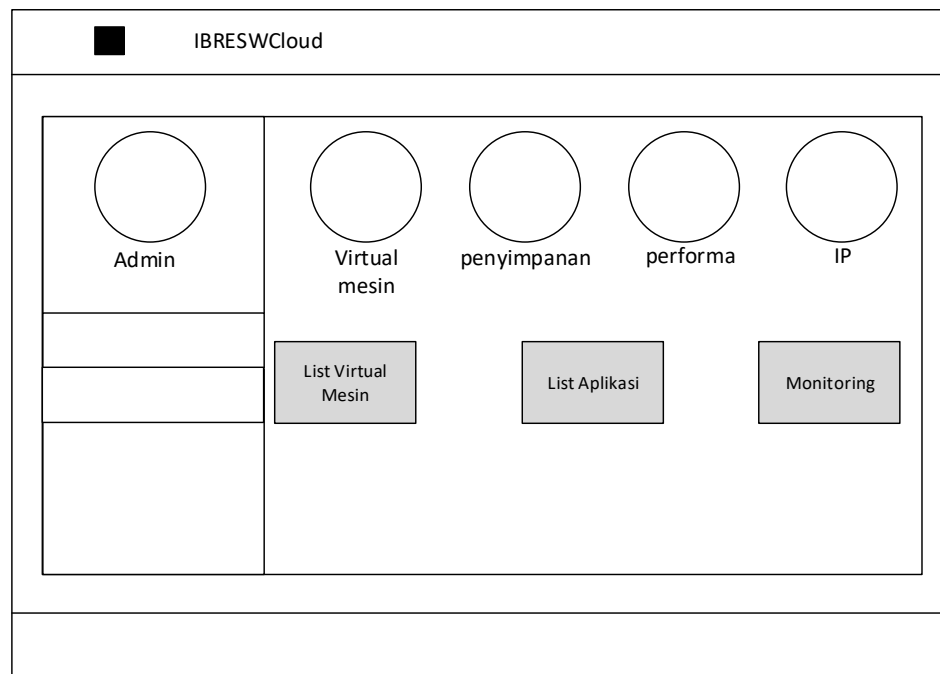
Pada tahap ini akan menjelaskan menu yang diperoleh *administrator* dalam sistem *PaaS*.



The image shows a login form titled "IBRESWCloud". Below the title is the text "Sign In". There are two input fields: "Username :" and "Password :". Below the password field is a "Login" button. At the bottom of the form, there are two links: "Register Now!" and "Forgot Password ?".

Gambar 7.6.2.1 Tampilan *form login administrator*

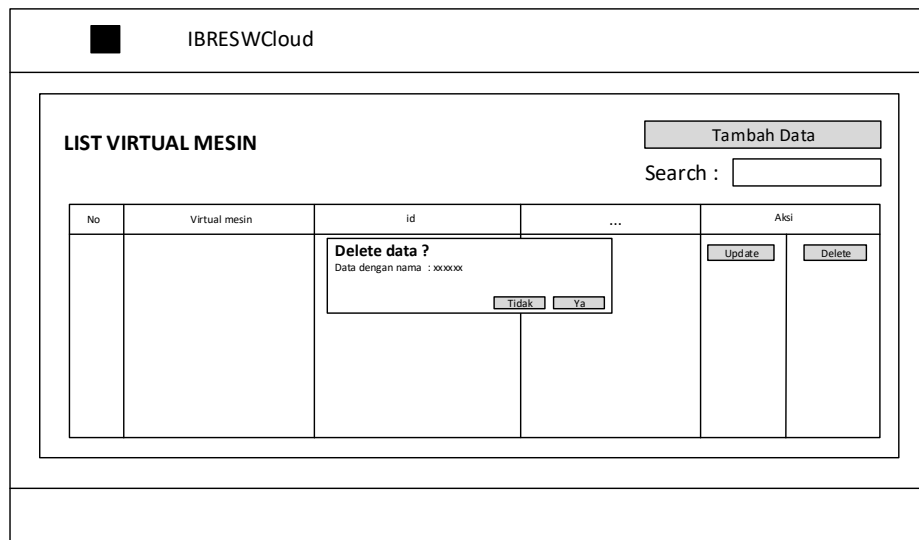
Untuk dapat masuk sistem administrator harus melakukan *login* pada *website*, *login administrator* sama seperti yang dilakukan oleh *user*, hanya saja hak akses yang dimiliki oleh *administrator* berbeda.



The image shows the administrator dashboard for "IBRESWCloud". The dashboard has a header with the logo and name "IBRESWCloud". The main content area is divided into two columns. The left column has a sidebar with a circular icon labeled "Admin" and three empty rectangular boxes below it. The right column has five circular icons labeled "Virtual mesin", "penyimpanan", "performa", and "IP". Below these icons are three rectangular buttons labeled "List Virtual Mesin", "List Aplikasi", and "Monitoring".


Gambar 7.6.2.2 Tampilan *dashboard administrator*

Setelah melakukan *login*, *administrator* akan dialihkan ke *dashboard* sistem seperti gambar 7.6.2.2. pada *dashboard* akan terlihat berapa *virtual* mesin yang ada, penyimpanan yang sudah digunakan, performa *CPU*, *IP public* yang sudah terpakai, serta *monitoring server*.



Gambar 7.6.2.3 Tampilan *list virtual* mesin *user*

Pada gambar 7.6.2.3. menampilkan *list virtual* mesin yang dibuat oleh *user*, pada bagian ini *administrator* dapat menghapus *virtual* mesin atau melakukan *update* bila *user* ingin mengganti sistem operasi.


IBRESWCloud

LIST APLIKASI

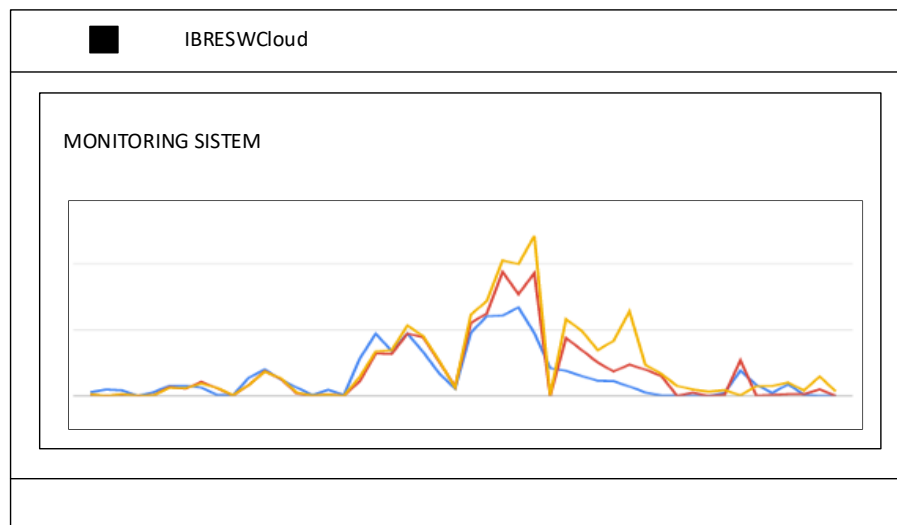
Tambah Data

Search :

No	Virtual mesin	id	...	Aksi
		<div> Delete data ?  Data dengan nama : xxxxxx  <div>Tidak Ya</div> </div>		<div>Update</div> <div>Delete</div>

Gambar 7.6.2.4 Tampilan *list* aplikasi tersedia

Pada gambar 7.6.2.4 merupakan *form* untuk menampilkan aplikasi yang dapat *diinstall* secara otomatis pada *virtual* mesin.



Gambar 7.6.2.5 Tampilan *menu monitoring*

Pada gambar 7.6.2.5 merupakan *menu* untuk melakukan *monitoring*. *Administrator* bertugas membuat *server* berjalan dengan baik serta menangani bila



terjadi sesuatu yang tidak diinginkan. Oleh karena itu penting dilakukan *monitoring* dari sisi *administrator*. Penerapan metode adaptif *threshold* untuk memberi ambang batas pada *load CPU* sehingga pada proses *monitoring* tersebut dapat mengetahui maksimal *CPU* bekerja pada *presentase* perhitungan yang dilakukan *threshold* adaptif.

### 7.7. Evaluasi Perancangan Sistem

Pada tahap ini dilakukan evaluasi terhadap desain perancangan sistem. Bila sistem telah sesuai dengan kebutuhan awal yang didefinisikan akan dilanjutkan ke tahap implementasi. Namun apabila desain sistem belum memenuhi kebutuhan awal yang didefinisikan, maka akan dilakukan perancangan ulang desain sistem.

## 8. Jadwal Pelaksanaan Penelitian

Jadwal pelaksanaan penelitian yang akan dilakukan dalam rangka menyelesaikan penelitian ini ditunjukkan pada Tabel 8.1.

Tabel 8.1 Rancangan Jadwal Pelaksanaan Penelitian

No.	Kegiatan	Minggu ke-											
		1	2	3	4	5	6	7	8	9	10	11	12
1	Studi Literatur												
2	Pengumpulan Data												
3	Perancangan Sistem												
4	Pembuatan Sistem												
5	Pengujian Sistem												
6	Penulisan Laporan Penelitian												

## DAFTAR PUSTAKA

- Anggeriana, H. (2011). *Cloud Computing*.
- Collings, T., & Kurt, W. (2015). *Duties of the System Administrator. In Red Hat Linux Networking System Administrator (chap. 1)*.
- Corporation, Exabyte. (2004). *The Basic Backup Guide*. cororado.
- Doty, S. (2008). *Python Basics*.
- Everett, D. G., & Jr, R. M. (2007). *Software Testing: Testing Across the Entire Software Development Life Cycle 1st Edition*. Canada: IEEE Press.
- Masse, M. (2012). REST API. Dalam *REST API Design Rulebook* (hal. 5). America: O'Reilly Media, Inc.
- Mulyana, E. (2017, 10 05). *Pengantar Openstack*. Diambil kembali dari <https://eueung.gitbooks.io>: <https://eueung.gitbooks.io/buku-komunitas-sdn-rg/content/index.html>
- Ng, N., Chang, H., Zou, Z., & Tang, S. (2011). An Adaptive Threshold Method to Address Routing Issues in Delay-Tolerant Networks.
- RedHat.Inc. (2017). *Ansible Documentation*. Diambil kembali dari [ansible.com](http://docs.ansible.com/ansible/latest/index.html): <http://docs.ansible.com/ansible/latest/index.html>
- Sosinsky, B. (2011). *Cloud Computing Bible*. Canada: Wiley Publishing, Inc.
- Wei, Y., Blake, M. B., & Saleh, I. (2013). Adaptive Resource Management for Service Workflows in Cloud Environments.
- Xia, Q., Lan, Y., & Xiao, L. (2015). A Heuristic Adaptive Threshold Algorithm on IaaS Clouds.
- Xiaojiang, L., & Yanlei, S. (2013). The Design and Implementation of Resource Monitoring for Cloud Computing Service Platform.
- Zuo, L., Shu, L., Dong, S., Zhou, Z., & Wang, L. (2015). A Dynamic Self-adaptive Resource-Load Evaluation Method in Cloud Computing.