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 buat 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.	*A Dynamic Self-adaptive Resource-Load Evaluation Method in Cloud Computing*	2015	Liyun Zuo, Lei Shu, Shoubin Dong, Zhangbing Zhou, Lei Wang
2.	*Adaptive Resource Management for Service Workflows in Cloud Environments*	2013	Yi Wei, M. Brian Blake dan Iman Saleh
3.	*A Heuristic Adaptive Threshold Algorithm on IaaS Clouds*	2015	Qingxin Xia dan Yuqing Lan, Limin Xiao
4.	*The Design and Implementation of Resource Monitoring for Cloud Computing Service Platform*	2013	Lei Xiaojiang, Shang Yanlei
5.	*An Adaptive Threshold Method to Address Routing Issues in Delay-Tolerant Networks*	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 algortima 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 Memanajemen 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 *troughput*, 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 abtraksi 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, penyimpan 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* mengunakan *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 anatara 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 ditambhakan 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 softaware 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 ε panjang beban kerja (α) , *CPU* (β) , penggunaan *memory* (γ) 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 Ti merupakan parameter Sumber daya virtual mesin dan *Si* 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, *Si* 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 metodelogi 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 :