ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

RANCANG BANGUN SISTEM PENENTUAN KEPUTUSAN UNTUK DISTRIBUSI PENYEDIAN KONTAINER DENGAN MULTI KRITERIA SECARA DINAMIS

DANIEL FABLIUS
NRP 5113 100 109

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

RANCANG BANGUN SISTEM PENENTUAN KEPUTUSAN UNTUK DISTRIBUSI PENYEDIAN KONTAINER DENGAN MULTI KRITERIA SECARA DINAMIS

DANIEL FABLIUS
NRP 5113 100 109

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

DESIGN AND IMPLEMENTATION OF DECISION MAKING SYSTEM FOR DYNAMIC CONTAINER PROVISIONING DISTRIBUTION WITH MULTI CRITERIA

DANIEL FABLIUS
NRP 5113 100 109

Supervisor I
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN SISTEM PENENTUAN KEPUTUSAN UNTUK DISTRIBUSI PENYEDIAN KONTAINER DENGAN MULTI KRITERIA SECARA DINAMIS

TUGAS AKHIR
Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
DANIEL FABLIUS
NRP: 5113 100 109

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD ..... (Pembimbing 1)
NIP: 197708242006041001

Bagus Jati Santoso, S.Kom., Ph.D ..... (Pembimbing 2)
NIP: 051100116

SURABAYA
Juni 2017

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN SISTEM PENENTUAN KEPUTUSAN UNTUK DISTRIBUSI PENYEDIAN KONTAINER DENGAN MULTI KRITERIA SECARA DINAMIS

Nama	:	**DANIEL FABLIUS**
NRP	:	**5113 100 109**
Jurusan	:	**Teknik Informatika FTIf**
Pembimbing I	:	**Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD**
Pembimbing II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstrak

Saat ini penggunaan kontainer docker dalam dunia tekonologi sangat banyak dilakukan. Kontainer docker merupakan operating-system-level virtualization untuk menjalankan beberapa sistem linux yang terisolasi (kontainer) pada sebuah host. Kontainer berfungsi untuk mengisolasi aplikasi atau servis dan dependensinya. Untuk setiap servis atau aplikasi yang terisolasi dibutuhkan satu kontainer pada server host yang ada dan setiap kontainer akan menggunakan sumber daya yang ada pada server host selama kontainer tersebut menyala. Oleh karena itu, jika servis atau aplikasi yang disediakan terus bertambah, maka kontainer juga akan terus bertambah. Hal ini akan menimbulkan masalah dikarenakan sumber daya server atau host yang terbatas.

Oleh karena itu diperlukan beberapa server untuk menyediakan sebuah servis atau aplikasi yang terus bertambah. Akan tetapi, ketika menggunakan beberapa server atau host untuk menjalankan servisnya, ketersediaan sumber daya pada setiap server seringkali berbeda-beda. Hal ini dapat menimbulkan masalah dalam pendistribusian kontainer pada

setiap server, karena jika pendistribusian penyedian kontainer tidak memperhatikan ketersediaan sumber daya pada setiap server, maka penyedian kontainer menjadi tidak efisien. Oleh karena itu, dibutuhkan sebuah cara untuk mengambil keputusan server manakah yang paling baik digunakan oleh pengguna pada saat permintaan penyediaan kontainer datang.

Dalam tugas akhir ini, akan digunakan salah satu metode Multi Criteria Decision Making (MCDM) yaitu Analytical Hierarchy Process sebagai metode pengambilan keputusan. Analytical Hierarchy Process (AHP) adalah sebuah metode pengambilan keputusan yang dilakukan berdasarkan beberapa paramater yang diambil dari sejumlah server yang berbeda. Parameter-parameter yang digunakan berupa ketersediaan sumber daya dari setiap server host seperti ketersediaan RAM atau memory, CPU, dan Penyimpanan file. Dari parameter-parameter tersebut akan diambil sebuah server yang akan melakukan penyediaan kontainer.

***Kata-Kunci**: Analytical Hierarchy Process (AHP), kontainer docker, multi kriteria.*

DESIGN AND IMPLEMENTATION OF DECISION MAKING SYSTEM FOR DYNAMIC CONTAINER PROVISIONING DISTRIBUTION WITH MULTI CRITERIA

Name	:	**DANIEL FABLIUS**
NRP	:	**5113 100 109**
Major	:	**Informatics FTIf**
Supervisor I	:	**Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD**
Supervisor II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstract

Currently, the use of docker container in the world of technology is very much done. Docker container is a operating-system-level virtualization to run some isolated linux system (container) on a host. Containers serve to isolate applications or services and their dependencies. For every isolated service or application, it takes one container on an existing host server and each container will use resource on the server host as long as the container is alive. Therefore, if the service or application provided continues to grow, then the container will also continue to grow. This will cause problems due to limited server or host resources.

Therefore it takes some servers to provide an ever-increasing service or application. However, when using multiple servers or hosts to run its services, the resources and performance of each server are often different. This can cause problems in the distribution of containers on each server, because if the distribution of container providers does not pay attention to resource availability and performance of each server, then the container supply becomes inefficient. Therefore, it takes a way to

decide which server is best used by the user upon request of container deployment arrives.

In this final project, we will use one of the Multi Criteria Decision Making (MCDM) method which is the Analytical Hierarchy Process as a decision-making method. Analytical Hierarchy Process is a decision-making method based on several parameters taken from a number of different servers. The parameters used can be either RAM or Memory,CPU or File Storage usage. From these parameters a server will be chosen which will contain the container.

***Kata-Kunci**: Analytical Hierarchy Process (AHP), container, multi criteria.*

KATA PENGANTAR

Segala puji bagi Tuhan Yesus Kristus, yang telah melimpahkan rahmat-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang Bangun Sistem Penentuan Keputusan untuk Distribusi Penyedian Kontainer dengan Multi Kriteria Secara Dinamis**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Tuhan Yesus Kristus, atas anugerah dan pencerahan-Nya yang tidak terkira kepada penulis.
2. Keluarga yang senantiasa memberikan do'a, dukungan, dan motivasi hingga saat ini.
3. Royyana Muslim Ijtihadie, S.Kom., M.Eng., Ph.D. selaku dosen pembimbing I yang telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Bagus Jati Santoso, S.Kom., Ph.D. selaku dosen pembimbing II yang telah meluangkan waktu untuk membimbing, memotivasi, dan membantu penulis ketika penulis kesulitan dalam mengerjakan Tugas Akhir.
5. Bapak Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS saat ini, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah memberikan ilmu dan pengalamannya.
6. Teman-teman Kontrakan Berprestasi, Daniel Bintar, M.Luthfie La Roeha, Dewangga, Donny, Irsyad, Juan, dan

Wicak yang telah membantu penulis dalam menyelesaikan permasalahan dalam mengerjakan tugas akhir, yang selalu menghibur, memotivasi, dan mendukung penulis untuk menyelesaikan tugas akhir ini.

7. Teman-teman Administrator AJK, Zaza, Nindy, Risma, Uul, Wicak, Asbun, Syukron, Fatih, Ambon, Oing, Vivi, Bebet,Thoni, Awan,Fuad, Didin dan satria yang bersedia direpotkan, merepotkan dan menemani penulis dalam masa pengerjaan tugas akhir ini.
8. Alumni-alumni, Mas Surya, Mas Thiar, Mas Romen, Mas Sam, Mas Uyung, Mas Agus, yang masih bersedia memberikan pencerahan selama pengerjaan tugas akhir ini.
9. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2017

Daniel Fablius

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Saat ini penggunaan kontainer *docker* dalam dunia tekonologi sangat banyak dilakukan. Kontainer *docker* merupakan *operating-system-level virtualization* untuk menjalankan beberapa sistem linux yang terisolasi (kontainer) pada sebuah *host* atau *server*. Kontainer berfungsi untuk mengisolasi aplikasi atau servis dan dependensinya.

Untuk setiap servis atau aplikasi yang terisolasi, dibutuhkan satu kontainer pada *server* host yang ada. Oleh karena itu, jika servis atau aplikasi yang disediakan terus bertambah, maka kontainer juga akan terus bertambah dan setiap kontainer akan menggunakan sumber daya yang ada pada *server* host selama kontainer tersebut menyala. Hal ini akan menimbulkan masalah dikarenakan sumber daya *server* atau host yang terbatas. Oleh karena itu diperlukan beberapa *server* untuk menyediakan sebuah servis atau aplikasi yang terus bertambah. Dengan adanya beberapa *server* yang digunakan sebagai host, pendistribusian kontainer akan diperlukan jika jumlah kontainer yang akan dipasang akan terus bertambah. Pendistribusian kontainer juga harus dilakukan secara dinamis, dengan kata lain, *server* atau *host* yang digunakan haruslah berubah-ubah sesuai dengan kondisi tertentu. Hal ini diperlukan agar tidak hanya salah satu *server* yang terbebani dengan penyediaan kontainer.

Namun, ketika menggunakan beberapa *server* atau host untuk menjalankan servisnya, spesifikasi setiap *server* seringkali berbeda-beda. Hal ini dapat menimbulkan masalah dalam

pendistribusian kontainer pada setiap *server*, karena jika pendistribusian penyediaan kontainer tidak memperhatikan spesifikasi setiap *server*, maka penyedian kontainer menjadi tidak efisien.

Oleh karena itu, dibutuhkan sebuah cara untuk memilih *server* yang paling baik untuk digunakan oleh pengguna pada saat permintaan penyediaan kontainer datang. Dalam proses pengambilan keputusan tersebut, ketersediaan sumber daya pada setiap *server* dapat dijadikan sebagai parameter yang akan digunakan sebagai acuan pada proses pengambilan keputusan. Dalam tugas akhir ini, akan digunakan salah satu metode *Multi Criteria Decision Making (MCDM)* yaitu *Analytical Hierarchy Process (AHP)* sebagai metode pengambilan keputusan untuk pendistribusian kontainer *docker* secara dinamis.

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah dalam tugas akhir ini:

1. Bagaimana cara mendistribusikan kontainer ke *server* yang ada berdasarkan multi kriteria secara dinamis?
2. Bagaimana cara menentukan *server* penyedia kontainer berdasarkan multi kriteria pada *server* penyedia kontainer?
3. Bagaimana performa sistem pendistribusian kontainer dengan metode pengambilan keputusan AHP dibandingkan dengan sistem dengan metode lain?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Algoritma pengambilan keputusan yang digunakan adalah metode *Analytical Hierarchy Process*.

2. Kontainer yang digunakan adalah *docker*.
3. Sistem memiliki koneksi SSH ke setiap *server* atau host penyedia kontainer *docker*.
4. Parameter untuk penentuan pengambilan keputusan distribusi *server* adalah penggunaan CPU, RAM dan Penyimpanan File.
5. Uji coba aplikasi akan menggunakan REST API.

1.4 Tujuan

Tugas akhir dibuat dengan beberapa tujuan. Berikut beberapa tujuan dari pembuatan tugas akhir:

1. Membuat sebuah sistem pendistribusian kontainer ke *server* berdasarkan multi kriteria secara dinamis.
2. Mengimplementasikan metode pengambilan keputusan untuk pendistribusian kontainer yang efisien berdasarkan penggunaan RAM, CPU dan Penyimpanan File pada *server*.

1.5 Manfaat

Tugas akhir dibuat dengan beberapa manfaat. Berikut beberapa manfaat dari pembuatan tugas akhir:

1. Menanggulangi masalah pendistribusian kontainer dengan multi kriteria secara dinamis.
2. Mempelajari performa sistem pendistribusian kontainer dengan metode pengambilan keputusan *Analytical Hierarchy Process* dibandingkan dengan metode pendistribusian lain.

1.6 Metodologi

Metodologi yang digunakan pada pengerjaan Tugas Akhir ini adalah sebagai berikut:

1.6.1 Studi literatur

Studi literatur merupakan langkah yang dilakukan untuk mendukung dan memastikan setiap tahap pengerjaan tugas akhir sesuai dengan standar dan konsep yang berlaku. Pada tahap studi literatur ini, akan dilakukan studi mendalam mengenai *Analytical Hierarchy Process (AHP)* , kontainer *docker*, pendistribusian beban kerja secara dinamis dan pengambilan keputusan berdasarkan multi kriteria. Adapun literatur yang dijadikan sumber berasal dari paper, buku, materi perkuliahan, forum serta artikel dari internet.

1.6.2 Desain dan Perancangan Sistem

Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep. Tahap ini merupakan tahap yang paling penting dimana bentuk awal aplikasi yang akan diimplementasikan didefinisikan. Pada tahapan ini dibuat kasus penggunaan yang ada pada sistem, arsitektur sistem, serta perencanaan implementasi pada sistem.

1.6.3 Implementasi Sistem

Implementasi merupakan tahap membangun implementasi rancangan sistem yang telah dibuat. Pada tahapan ini merealisasikan apa yang telah didesain dan dirancang pada tahapan sebelumnya, sehingga menjadi sebuah sistem yang sesuai dengan apa yang telah direncanakan.

1.6.4 Uji Coba dan Evaluasi

Pada tahapan ini dilakukan uji coba terhadap sistem yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Selain itu, tahap ini juga akan melakukan uji performa sistem dan melakukan

perbandingan dengan metode lain untuk mengetahui efisiensi penggunaan sumber daya serta evaluasi berdasarkan hasil uji performa tersebut.

1.7 Sistematika Laporan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna bagi pembaca yang berminat melakukan pengambangan lebih lanjut. Secara garis besar, buku tugas akhir ini terdiri atas beberapa bagian seperti berikut:

1. **Bab I Pendahuluan**
   Bab yang berisi latar belakang, tujuan, manfaat, permasalahan, batasan masalah, metodologi yang digunakan dan sistematika laporan.

2. **Bab II Dasar Teori**
   Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang yang digunakan dalam pembuatan tugas akhir ini.

3. **Bab II Desain dan Perancangan**
   Bab ini berisi tentang analisis dan perancangan sistem yang dibuat, termasuk di dalamnya mengenai analisis kasus penggunaan, desain arsitektur sistem, dan perancangan implementasi sistem.

4. **Bab IV Implementasi**
   Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa pemasangan alat dan kode program yang digunakan untuk mengimplementasikan sistem.

5. **Bab V Uji Coba dan Evaluasi**
   Bab ini membahas tahap-tahap uji coba serta melakukan evaluasi terhadap sistem yang dibuat.

6. **Bab VI Kesimpulan dan Saran**
   Bab ini merupakan bab terakhir yang memberikan kesimpulan dari hasil percobaan dan evaluasi yang telah dilakukan. Pada bab ini juga terdapat saran bagi pembaca yang berminat untuk melakukan pengembangan lebih lanjut.

BAB II

LANDASAN TEORI

2.1 Analytical Hierarchy Process

AHP atau *Analytical Hierarchy Process* adalah teknik terstruktur untuk menangani keputusan kompleks berdasarkan matematika dan psikologi. AHP dikembangkan oleh Thomas L. Saaty pada tahun 1970an dan telah dipelajari dan disempurnakan sejak saat itu. AHP menyediakan kerangka kerja yang komprehensif dan rasional untuk menyusun suatu masalah keputusan, untuk mewakili dan mengkuantifikasi elemen-elemennya, untuk menghubungkan elemen-elemen tersebut dengan tujuan keseluruhan, dan untuk mengevaluasi solusi alternatif.

AHP digunakan di seluruh dunia dalam berbagai situasi pengambilan keputusan, di bidang seperti pemerintah, bisnis, industri, perawatan kesehatan dan pendidikan. AHP mengembangkan prioritas untuk alternatif yang berbeda dan berdasarkan kriteria yang ada akan diambil sebuah keputusan untuk alternatif tersebut. Prioritas awalnya ditetapkan sesuai kepentingan untuk mencapai tujuan, setelah itu prioritas ditetapkan juga untuk performa dari alternatif pada setiap kriteria, prioritas ini ditetapkan berdasarkan penilaian berpasangan dengan menggunakan pengambilan keputusan, atau pengukuran dari suatu skala jika ada[1]. Satu skala umum (diadaptasi dari Saaty) ditunjukkan pada Tabel 2.1

Tabel 2.1: Daftar Skala Prioritas pada AHP

Tingkat Kepentingan	**Definisi**	**Penjelasan**
1	Sama Penting	2 faktor berkontribusi senilai terhadap objektif yang ada.

Tabel 2.1: Daftar Skala Prioritas pada AHP

Tingkat Kepentingan	**Definisi**	**Penjelasan**
3	Sedikit Lebih Penting	salah satu faktor lebih penting sedikit dibandingkan faktor yang lain.
5	Lebih Penting	salah satu faktor lebih penting dibandingkan faktor yang lain.
7	Sangat Lebih Penting	salah satu faktor sangat lebih penting dibandingkan faktor yang lain.
9	Benar-benar Lebih Penting	Bukti yang mendukung salah satu faktor dari yang lain telah mencapai kemungkinan yang tertinggi.
2,4,6,8	Nilai Tengah	Nilai saat dimana dibutuhkan kompromi.

Prosedur dasar untuk melaksanakan AHP terdiri dari langkah-langkah berikut:

1. **Penataan permasalahan dan pemilihan kriteria** Langkah pertama adalah menguraikan masalah pengambilan keputusan menjadi bagian-bagian penyusunnya. Dalam bentuk yang paling sederhana, permasalahan terdiri dari tujuan atau fokus permasalahan pada tingkat paling atas, kriteria (dan subkriteria) pada tingkat menengah, sedangkan tingkat terendah berisi pilihannya. Mengatur semua komponen-kompenen

tersebut dalam sebuah hierarki memberikan gambaran keseluruhan tentang hubungan yang kompleks antar kompenen dan membantu pengambil keputusan untuk menilai apakah elemen di setiap tingkat memiliki tingkat kepentingan yang sama sehingga dapat dibandingkan secara akurat. Elemen pada tingkat tertentu tidak harus berfungsi sebagai kriteria untuk semua elemen yang pada tingkat yang lebih rendah. Setiap tingkat dapat mewakili bagian yang berbeda dari masalah sehingga hierarki yang ada tidak perlu terbentun secara lengkap. Saat membangun hierarki, penting untuk mempertimbangkan lingkungan sekitar masalah dan untuk mengidentifikasi masalah atau atribut yang berkontribusi terhadap solusi dan juga untuk mengidentifikasi semua patisipan yang terkait dengan masalah tersebut.

2. **Pengaturan prioritas kriteria dengan perbandingan berpasangan (weighing)**
   Untuk masing-masing pasangan kriteria, pembuat keputusan diharuskan untuk menjawab pertanyaan seperti ”Seberapa penting kriteria A relatif terhadap kriteria B?” Menilai prioritas ”relatif” setiap kriteria dilakukan dengan menetapkan bobot antara 1 Dan 9 seperti pada tabel 2.1 terhadap kriteria yang lebih penting, sedangkan nilai timbal balik dari nilai ini akan diberikan pada pasangan dari kriteria tersebut. Pembobotan ini kemudian akan dinormalisasi untuk mendapatkan bobot untuk setiap kriteria.
3. **Perbandingan berpasangan terhadap pilihan pada setiap kriteria (scoring)**
   Untuk masing-masing pasangan dalam setiap kriteria, pilihan yang lebih baik akan diberikan nilai, sekali lagi, pada skala antara 1 dan 9, sementara pilihan lain pasangannya akan diberi nilai sama dengan nilai timbal

balik dari nilai ini. Setiap nilai akan menunjukkan seberapa baik pilihan ”x” untuk kriteria ”Y”. Setelah itu, peringkat akan dinormalisasi.

4. **Mendapatkan skor keseluruhan untuk setiap pilihan** Pada langkah terakhir, nilai dari setiap pilihan digabungkan dengan bobot kriteria untuk menghasilkan nilai keseluruhan untuk setiap pilihan. Sejauh mana pilihan memenuhi kriteria akan diukur sesuai dengan seberapa penting kriteria tersebut. Hal ini dilakukan dengan penjumlahan bobot sederhana.

2.2 *Node* JS

Node.js - juga disebut *node* - adalah lingkungan JavaScript sisi *server* (lihat http://nodejs.org). *Nodejs* berbasis pada *Google's runtime implementation* - bernama ”V8” *engine*. V8 dan *node* sebagian besar diimplementasikan dengan C dan C ++, yang berfokus pada performa dan konsumsi memori rendah. Tapi, sebagaimana VB sebagian besar mendukung JavaScript pada browser (terutama, Google Chrome), *node* bertujuan untuk mendukung proses pada *server* yang berjalan lama.

Tidak seperti kebanyakan lingkungan modern lainnya, proses *node* tidak bergantung pada *multithreading* untuk mendukung pelaksanaan proses bisnis secara bersamaan. Ini didasarkan pada model penjadwalan I/O asinkron. Proses *node* pada *server* dibayangkan sebagai proses *single-threaded daemon* yang menyematkan mesin JavaScript untuk mendukung kustomisasi. Hal ini berbeda dengan kebanyakan sistem penjadwalan untuk bahasa pemrograman lainnya, yang datang dalam bentuk *library*.

JavaScript sangat sesuai untuk pendekatan ini karena mendukung *event callback*. Misalnya, ketika browser memuat sebuah dokumen secara keseluruhan, pengguna mengeklik tombol, atau ketika permintaan Ajax terpenuhi, sebuah *event*

akan memicu *callback*. Sifat fungsional JavaScript membuatnya sangat mudah untuk membuat fungsi objek anonim yang dapat didaftarkan sebagai *event handler*.[2]

2.3 Ansible

Ansible adalah mesin otomatisasi *open source* yang mengotomatisasi penyediaan perangkat lunak, manajemen konfigurasi, dan pemasangan aplikasi[3]. Ansible disertakan sebagai bagian dari distro Fedora Linux, yang dimiliki oleh Red Hat Inc., dan juga tersedia untuk Red Hat Enterprise Linux, CentOS, dan Scientific Linux melalui Paket Ekstra untuk Enterprise Linux (EPEL) dan juga untuk sistem operasi lain. Seperti kebanyakan manajemen konfigurasi perangkat lunak, Ansible memiliki dua tipe *server*: mesin pengontrol dan *node*. Pertama, terdapat satu mesin pengontrol tunggal dimana mesin ini akan melakukan orkestrasi atau mengatur setiap node yang ada. *Node* dikelola oleh mesin pengontrol melalui SSH. Mesin pengontrol memetakan lokasi dari setiap *node* melalui *"inventory"*. Untuk mengatur *node*, Ansible menyebarkan modul-modul ke *node* yang ada melalui SSH. Modul-modul ini disimpan sementara di setiap *node* dan berkomunikasi dengan mesin pengontrol melalui protokol JSON. Ketika Ansible tidak sedang mengelola *node*, ia tidak mengkonsumsi sumber daya karena tidak ada daemon atau program yang dijalankan untuk Ansible di latar belakang. Berbeda dengan manajemen konfigurasi perangkat lunak populer - seperti Chef, Puppet, dan CFEngine - Ansible menggunakan arsitektur tanpa agen. Dengan arsitektur berbasis agen, *node* harus memiliki daemon terpasang secara lokal yang berkomunikasi dengan mesin pengendali. Dengan arsitektur tanpa agen, *node* tidak diharuskan memasang dan menjalankan daemon untuk terhubung dengan mesin pengendali. Jenis arsitektur ini mengurangi overhead pada

jaringan dengan mencegah *node* untuk terus mengecek apakah mesin pengendali telah siap atau tidak.

2.4 Collectd

Collectd adalah daemon Unix yang mengumpulkan, mentransfer dan menyimpan data kinerja komputer dan peralatan jaringan[4]. Data yang diperoleh dimaksudkan untuk membantu sistem administrator mempertahankan gambaran umum mengenai sumber daya yang ada untuk mendeteksi *bottlenecks* yang ada. Collectd menggunakan desain modular dimana daemon itu sendiri hanya mengimplementasikan infrastruktur untuk penyaringan dan penyampaian data serta fungsi tambahan dan memerlukan sumber daya yang sangat sedikit, selain itu daemon tersebut berjalan pada perangkat *embedded* bertenaga *OpenWrt*. Akuisisi dan penyimpanan data ditangani oleh *plug-in* dalam bentuk objek. Dengan cara ini kode yang spesifik untuk satu sistem operasi, sebagian besar dijauhkan dari daemon yang sebenarnya. *Plugin* dapat memiliki *dependency*nya sendiri, misalnya sistem operasi atau *library* tertentu. Tugas lain yang dilakukan oleh *plug-in* termasuk pemrosesan ”notifikasi” dan membuat log. *plug-in* Akuisisi data, yang disebut *read plug-in* dalam dokumentasi collectd, dapat secara kasar dimasukkan ke dalam tiga kategori:

1. *Plugin* sistem operasi mengumpulkan informasi seperti penggunaan CPU, penggunaan memori, atau jumlah pengguna yang masuk ke sistem. *Plugin* ini biasanya perlu di sesuaikan ke setiap sistem operasi. Tidak semua plug-in seperti ini tersedia untuk semua sistem operasi.
2. *Plugin* aplikasi mengumpulkan data kinerja dari atau tentang aplikasi yang berjalan pada komputer yang sama atau remote, misalnya Apache HTTP *server*. *Plugin* ini biasanya menggunakan *library* pada perangkat lunak.

3. *Plugin generic* menawarkan fungsi dasar yang dapat digunakan pengguna untuk melakukan tugas tertentu. Contohnya adalah *query* peralatan jaringan dengan menggunakan SNMP atau eksekusi program atau skrip.

"*write plug-in*" menawarkan kemungkinan untuk menyimpan data yang terkumpul di disk menggunakan file RRD atau CSV, atau untuk mengirim data melalui jaringan ke "*remote instance*" dari daemon.

2.5 InfluxDB

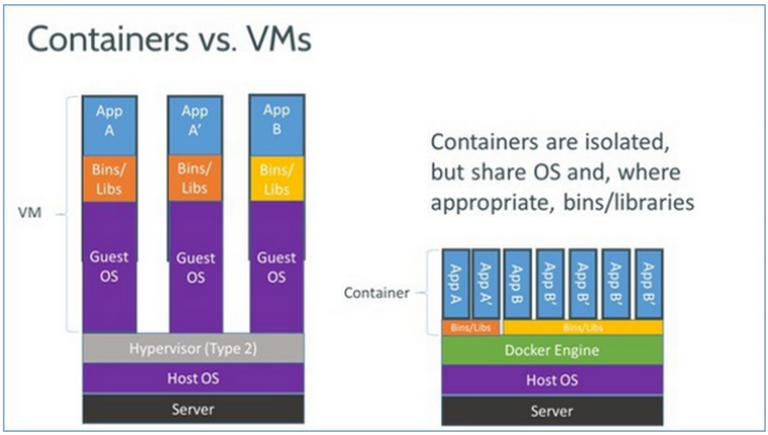
InfluxDB adalah basis data *time series open-source* yang dikembangkan oleh InfluxData. InfluxDB menerima data via HTTP, TCP, dan UDP[5]. InfluxDB ditulis pada bahasa Go dan dioptimalkan untuk penyimpanan dan pengumpulan data *time series* yang cepat dan memiliki ketersediaan tinggi seperti pemantauan operasi, metrik aplikasi, data sensor *Internet of Things*, dan analisis *real-time*. InfluxDB juga memiliki dukungan untuk pengolahan data dari Graphite atau Collectd. InfluxDB tidak memiliki dependensi eksternal dan menyediakan bahasa mirip SQL dengan fungsi *built-in time-centric* untuk *query* struktur data yang terdiri dari *measurements*, *series*, dan *points*. Setiap *point* terdiri dari beberapa pasangan *key-value* yang disebut *fieldset* dan *timestamp*. Saat dikelompokkan bersama oleh sekumpulan pasangan *key-value* hal ini akan disebut *tagset*, hal inilah yang menentukan sebuah *series*. Akhirnya, *series* dikelompokkan bersama oleh sebuah "*identifier*" *string* untuk membentuk *measurement*. Nilai dari setiap data dapat berupa bilangan bulat 64-bit, floating point 64-bit, *string*, dan *boolean*. *Point* diindeks oleh waktu dan *tagset* mereka. Kebijakan retensi didefinisikan pada *measurement* dan kontrol bagaimana data diturunkan dan dihapus. *Continuous queries* akan berjalan secara berkala, menyimpan hasilnya dalam *target measurement*.

2.6 Kontainer *Docker*

Docker adalah proyek *open source* yang menyediakan cara sistematis untuk mengotomatisasi penyebaran aplikasi Linux lebih cepat di dalam wadah portabel. Pada dasarnya, *docker* memperluas LXC dengan kernel dan API tingkat aplikasi yang bersama-sama menjalankan proses dalam isolasi: CPU, memori, I / O, jaringan, dan sebagainya. *Docker* juga menggunakan *namespace* untuk benar-benar mengisolasi tampilan aplikasi dari lingkungan operasi yang mendasarinya, termasuk *process tree*, jaringan, ID pengguna, dan sistem file. Kontainer *docker* dibuat menggunakan sebuah *image*. *Image docker* hanya bisa mencakup dasar-dasar OS, atau dapat terdiri dari satu set aplikasi *prebuilt* yang siap dijalankan. Saat membuat *image* dengan *docker*, setiap perintah yang dijalankan (yaitu perintah dijalankan, seperti apt-get install) membentuk lapisan baru di atas lapisan sebelumnya. Perintah dapat dijalankan secara manual atau otomatis menggunakan *dockerfiles* [6].

Setiap *dockerfile* adalah skrip yang terdiri dari berbagai perintah (instruksi) dan argumen yang terdaftar secara berturut-turut untuk secara otomatis melakukan tindakan pada *image* dasar untuk membuat (atau membentuk) *image* baru. Mereka digunakan untuk mengatur penempatan artefak dan menyederhanakan proses penyebaran dari awal sampai selesai. Kontainer juga bisa berjalan di *VMs*. Jika *cloud* memiliki runtime kontainer asli yang benar, sebuah kontainer dapat berjalan langsung di VM. Jika *cloud* hanya mendukung VM berbasis *hypervisor*, maka tidak akan ada masalah - keseluruhan aplikasi, kontainer, dan *OS stack* dapat ditempatkan pada VM dan dijalankan seperti aplikasi lainnya. *Docker* dapat dijalankan di berbagai sistem operasi, pengembang dapat dengan mudah menggunakan layanan *docker* melalui `https://hub.docker.com` untuk mengunduh *images* yang

diinginkan. Perbedaan antara kontainer dan *Virtual Machine* ditunjukkan pada Gambar 2.1



Gambar 2.1: Perbandingan *docker* dan virtual machine[6]

(Halaman ini sengaja dikosongkan)

BAB III

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan dari sistem.

3.1 Deskripsi Umum Sistem

Sistem yang akan dibuat adalah sebuah sistem yang dapat menentukan sebuah *server* yang akan menyediakan kontainer dengan aplikasinya yang disebut *docker host* terbaik berdasarkan ketersediaan sumber daya seperti ketersediaan CPU, memori dan penyimpanan file pada masing-masing *docker host*. Setelah itu, berhasil menentukan *server* terbaik, sistem akan mengirimkan perintah penyediaan kontainer *docker* ke *docker host* yang telah dipilih. Data-data sumber daya setiap *docker host* yang ada akan dikirimkan kepada suatu database berbasis deret waktu yang terdapat pada *server* pengambilan keputusan setiap lima detik sekali.

Setiap permintaan penyediaan kontainer *docker* dari user akan diarahkan ke *middleware* yang berisi sebuah webservice yang akan melakukan proses pengambilan keputusan untuk menentukan *docker host* mana yang akan di pilih untuk menyediakan kontainer *docker* yang diinginkan.

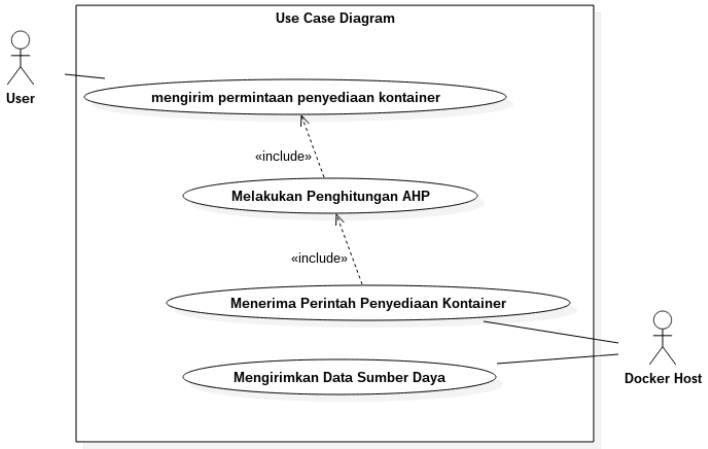
Proses pengambilan keputusan akan diawali dengan melihat jumlah *docker host* yang tersedia dan *hostname* dari setiap *docker host* yang sudah terdaftar pada suatu basis data. Kemudian *middleware* akan mengecek ketersediaan sumber daya pada setiap *docker host* tersebut yang telah terkumpul pada database *time series*. Setelah itu nilai-nilai ketersedian sumber daya pada setiap *docker host* akan diambil dan dihitung menggunakan algoritma *Analytic Hierarchy Process (AHP)* untuk menghasilkan *hostname* dari *docker host* yang terbaik untuk dipasangkan kontainer *docker* beserta dengan aplikasinya.

Setelah hostname *docker host* terbaik telah dipilih,

middleware akan melakukan proses remote ke *docker host* pilihan menggunakan protokol ssh dan memasangkan kontainer *docker* dengan aplikasinya pada *docker host* tersebut.

3.2 Kasus Penggunaan

Terdapat dua aktor dalam sistem yang akan dibuat yaitu User dan *Docker Host*. User adalah aktor yang melakukan permintaan penyediaan kontainer, sedangkan *docker host* adalah aktor yang akan menjadi tempat penyedia kontainer dan menerima perintah penyediaan kontainer. Diagram kasus penggunaan menggambarkan kebutuhan-kebutuhan yang harus dipenuhi sistem. Diagram kasus penggunaan digambarkan pada Gambar 3.1.



Gambar 3.1: Digram Kasus Penggunaan

Digram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.1.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-0001	Mengirim Permintaan Penyediaan Kontainer	User dapat mengirimkan permintaan penyediaan kontainer pada web service yang ada.
UC-0002	Melakukan Penghitungan AHP	Proses dimana setelah permintaan penyediaan kontainer diterima, sistem akan melakukan proses AHP untuk memilih *docker host* untuk menyediakan kontainer.
UC-0003	Menerima Perintah Penyediaan Kontainer	Proses dimana *docker host* akan menerima perintah dari sistem, untuk menyediakan kontainer.
UC-0004	Mengirim Data Resource	*docker host* dapat mengirim data resource yang ada pada saat ini untuk dijadikan acuan penghitungan AHP.

3.3 Arsitektur Sistem

Pada Sub-bab ini, dibahas mengenai tahap analisis arsitektur, analisis teknologi dan desain sistem yang akan dibangun.

3.3.1 Desain Umum Sistem

Berdasarkan deskripsi umum sistem yang telah ditulis diatas, dapat diperoleh kebutuhan sistem ini, diantaranya :

1. Pengumpulan data ketersediaan sumber daya *docker host*.
2. Penyimpanan daat ketersediaan sumber daya *docker host*.
3. Pengambil keputusan *docker host* terbaik untuk penyediaan kontainer *docker* menggunakan AHP.
4. Pemasangan kontainer *d*ocker pada *docker host* pilihan.

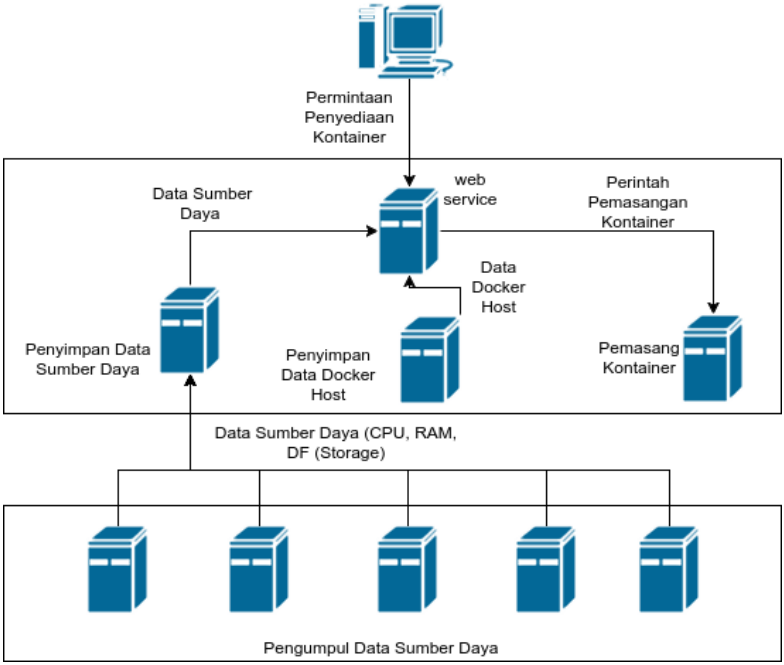
Untuk memenuhi kebutuhan sistem tersebut, penulis membagi sistem menjadi beberapa komponen. Komponen yang akan dibangun antara lain:

1. Pengumpul Data Sumber Daya *docker host*
   Berfungsi agar sistem mengambil data ketersediaan sumber daya pada masing-masing *docker host*.
2. Penyimpanan Data Sumber Daya *docker host*
   Berfungsi penyimpan data sumber daya *docker host* yang telah berhasil di kumpulkan, yang digunakan sebagai acuan pengambilan keputusan dengan AHP.
3. *Middleware*
   Berfungsi sebagai penerima permintaan penyediaan kontainer dari user dan juga berfungsi untuk melakukan pengambilan keputusan menggunakan algoritma AHP untuk menentukan *docker host* terbaik sebagai penyedia kontainer dan juga menghasilkan perintah untuk memasangkan kontainer ke *docker host* yang dipilih . Algoritma AHP yang digunakan akan mengambil data dari penyimpanan data yang berisi data ketersediaan sumber daya dari setiap *docker host* yang telah di kumpulkan

sebelumnya.

4. Pemasangan Kontainer
   Berfungsi untuk memasangkan kontainer *docker* pada *docker host* yang telah dipilih. Hal ini dilakukan dengan menjalankan sebuah perintah penyediaan kontainer yang dihasilkan pada middleware dan akan dijalankan pada *docker host* yang telah dipilih.

Pada pada Gambar 3.2 ditujunkkan arsitektur sistem secara umum dengan detail-detail dari kompenen yang terdapat didalamnya. Setiap komponen tersebut akan diimplementasikan dengan teknologi pendukung yang dibutuhkan.



Gambar 3.2: Arsitektur Komponen Sistem

3.3.2 Perancangan Pengumpul Data Sumber Daya *docker host*

Pengumpul Data Sumber Daya *docker host* adalah komponen yang bertugas mengumpulkan data ketersediaan sumber daya pada setiap *docker host* yang akan dijadikan data acuan untuk penghitungan dalam algoritma AHP. Komponen ini akan ditempatkan pada setiap *docker host* yang akan dijadikan sebagai penyedia kontainer *docker* nantinya. Komponen ini nantinya akan berjalan sebagai proses di balik layar yang terus mengecek ketersediaan sumber daya pada setiap *docker host* dan data yang dikumpulkan akan di kirimkan ke suatu penyimpanan data. Data-data sumber daya *docker host* yang di pakai sebagai acuan penghitungan algoritma AHP meliputi data ketersediaan memori atau RAM, CPU dan penyimpanan data. Data-data sumber daya ini dipilih dikarenakan setiap kontainer *docker* yang akan di pasangkan pada setiap *docker host* akan menggunakan sumber daya tersebut.

3.3.3 Perancangan Penyimpanan Data Sumber Daya *docker host*

Penyimpanan data sumber daya *docker host* adalah komponen pada sistem yang berfungsi untuk menyimpan data-data sumber daya setiap *docker host* yang sudah dikumpulkan sebelumnya. Tempat penyimpanan ini akan terpisah dari *docker host* yang ada dan data yang ditempatkan pada penyimpanan ini akan digunakan untuk penghitungan AHP nantinya. Penyimpanan data yang digunakan adalah sebuah basis data berbasis deret waktu. Nilai-nilai data yang dikumpulkan nantinya akan diidektifikasi berdasarkan waktu data itu didapatkan. Basis data dengan basis deret waktu ini dipilih untuk memudahkan pengolahan data, yaitu dengan mengambil data berdasarkan interval waktu yang ditentukan atau mengambil data berdasarkan waktu terakhir data dicatat.

3.3.4 Perancangan *Middleware*

Menurut Oxford Dictionaries[oxford], Middleware (dalam istilah komputer) adalah suatu perangkat lunak yang menjadi penjembatan antara sistem operasi, basis data dan aplikasi dalam sebuah jaringan. Dalam desain umum sistem komponen middleware berfungsi sebagai penerima permintaan user untuk penyediaan kontainer *docker*, mengambil data jumlah *docker host* yang tersedia, mengambil data ketersediaan sumber daya setiap *docker host* dan menentukan *docker host* terbaik melalui algoritma AHP. Dikarenakan ada beberapa kebutuhan yang harus dipenuhi , komponen middleware dibagi lagi 2 sub kompoenen, yaitu:

1. Basis Data
   Berfungsi sebagai tempat penyimpanan data jumlah *docker host* yang tersedia sebagai penyedia kontainer *docker*.
2. *Web Service*
   Web Service berfungsi sebagai penerima permintaan dari user. Selain itu *Web Service* juga berfungsi untuk mengolah data dari basis data yang menyimpan data ketersediaan sumber daya *docker host* dengan menggunakan algoritma AHP, yang nantinya menghasilkan hostname *docker host* yang dipilih sebagai penyedia kontainer *doker*.

Pada tugas akhir ini, bahasa javascript dipilih sebagai bahasa pemrogramman yang digunakan untuk mengimplementasikan komponen middleware. Kode javascript yang digunakan akan dijalankan pada sisi *docker host* dengan bantuan *nodeJS*. NodeJS dan Javascript dipilih dikarenakan sifat dari *nodeJS* yaitu *Non-blocking asynchronous I/O*, sehingga bermafaat untuk mengatasi permintaan user kepada web service secara bersamaan dengan lebih baik. Lalu, pada bagian penyimpanan data *docker host* yang tersedia, Mysql dipilih sebagai RDBMS untuk tugas akhir ini.

3.3.4.1 Desain Basis Data

Komponen basis data berfungsi sebagai tempat penyimpanan data *docker host* yang tersedia. Dalam basis data ini terdapat satu entitas, yaitu *server*, yang berfungsi menyimpan data-data *server* yang tersedia sebagai penyedia kontainer *docker*. Terdapat beberapa atribut yang dibutuhkan *server*, ditunjukkan pada Tabel 3.2.

Tabel 3.2: Atribut basis Data

Nama	**Tujuan**
ID	ID *server*
Name	Hostname *server*

3.3.4.2 Desain Web Service

Komponen Web Service berfungsi untuk menerima permintaan penyediaan kontainer dari pengguna. Pada Webservice akan terdapat antarmuka atau rute yang akan menerima permintaan pengguna dengan parameter nama dari kontainer yang akan dibuat. Pada rute tersebut proses penghitungan AHP akan dilakukan terhadap permintaan pengguna untuk menghasilkan *docker host* yang akan dipilih untuk dipasangkan kontainer *docker*.

3.3.5 Perancangan Pemasangan Kontainer

Pemasangan Kontainer adalah komponen yang berfungsi untuk memasangkan kontainer yang di minta oleh user pada *docker host* yang telah dipilih melalui proses AHP. Proses ini dilakukan dengan cara melakukan remote menuju *server* pilihan melalui protokol ssh.

BAB IV

IMPLEMENTASI

Setelah melewati proses perancangan mengenai sistem yang akan dibuat, maka akan dilakukan implementasi dari sistem tersebut. Bab ini akan membahas mengenai implementasi dari sistem yang meliputi proses pembuatan setiap komponen sehingga sistem dapat berjalan dengan baik. Masing-proses pembuat komponen akan dilengkapi dengan *pseudocode* atau konfigurasi dari sistem.

4.1 Lingkungan Implementasi

Dalam mengimplementasikan sistem, digunakan beberapa perangkat pendukung sebagai ebrikut.

4.1.1 Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Komputer dengan *processor* Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz dan RAM 2GB
2. Dua Komputer dengan *processor* Intel(R) Core(TM)2 Duo CPU E7200 @ 2.53GHz dan RAM 2GB

4.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Sistem Operasi Ubuntu 16.04 LTS 64 Bit
2. Sistem Operasi Ubuntu 14.04 LTS 64 Bit
3. *NodeJS* versi 6.10.2 untuk pengembangan web service.
4. ExpressJS versi 4.15.2 sebagai Kerangka Kerja *NodeJS*.
5. MySQL versi 5.7.18 untuk Sistem Manajemen Basis Data.
6. InfluxDB versi 0.13.0 untuk Sistem Manajemen Basis Data Berbasi Deret Waktu.

7. Ansible versi 3.0.0 untuk Manajemen Pemasangan kontainer *docker* pada *server*.
8. Collectd versi 5.6.2.35 untuk Pengumpul data sumber daya pada setiap *server*.
9. NPM versi 3.10.10 untuk Manajemen Paket Installasi untuk pengembangan pada platform web service. item ExpressJS versi 4.15.2 sebagai Kerangka Kerja *NodeJS*.
10. MySQL versi 5.7.18 untuk Sistem Manajemen Basis Data.
11. InfluxDB versi 0.13.0 untuk Sistem Manajemen Basis Data Berbasi Deret Waktu.
12. *Docker* sebagai kontainer yang akan di pasangkan pada *server*.

4.2 Implementasi Pengumpul Data Ketersediaan Sumber Daya *Docker Host*

Pengumpul Data yang digunakan pada sistem ini adalah perangkat lunak Collectd versi 5.6.2.35. Diperlukan beberapa tahap untuk dapat menggunakan Collectd, yaitu tahap pemasangan dan konfigurasi. Untuk melakukan pemasangan Collectd versi 5.6 pada sistem operasi Ubuntu tambahkan terlebih dahulu perintah ”deb http://pkg.ci.collectd.org/deb trusty collectd-5.6” ke dalam file /etc/apt/sources.list.d/pkg.ci.collectd.org.list, lalu jalankan *command* pada terminal seperti Kode Sumber 4.1 .

```
sudo apt-get update
sudo apt-get install collectd
```

Kode Sumber 4.1: Command untuk installasi Colectd

Lalu, agar Colectd dapat mengumpulkan data-data sumber daya pada *docker host* seperti CPU, Memori/RAM dan Penyimpanan File, perlu dilakukan konfigurasi tambahan dengan menambahkan Kode Sumber 4.2 pada file `/etc/collectd/collectd.conf`.

```
LoadPlugin network
LoadPlugin cpu
LoadPlugin df
LoadPlugin memory
<Plugin network>
    \textit{server} "10.151.36.37" "25826"
</Plugin>
<Plugin cpu>
      ReportByCpu true
      ReportByState true
      ValuesPercentage true
</Plugin>
<Plugin df>
      Device "/dev/sda6"
      MountPoint "/"
      FSType "ext4"
      IgnoreSelected false
      ReportInodes false
      ValuesAbsolute true
      ValuesPercentage true
</Plugin>
<Plugin memory>
        ValuesAbsolute true
        ValuesPercentage true
</Plugin>
```

Kode Sumber 4.2: Konfigurasi tambahan Collectd

Setelah itu, jalankan Kode Sumber 4.3 untuk mengaktifkan konfigurasi yang baru saja dibuat.

```
sudo service collectd restart
```

Kode Sumber 4.3: Command untuk menjalankan Collectd

4.3 Implementasi Penyimpanan Data Ketersediaan Sumber Daya *Docker Host*

Penyimpanan data yang digunakan pada sistem ini adalah sistem data berbasis deret waktu, InfluxDB. Diperlukan beberapa tahap untuk dapat menggunakan InfluxDB, yaitu tahap pemasangan dan konfigurasi. Untuk melakukan pemasangan InfluxDB versi 0.13.0 pada sistem operasi Ubuntu unduh terlebih

dahulu paket deb dari `https://dl.influxdata.com/influxdb/releases/influxdb_0.13.0_amd64.deb` lalu jalankan Kode Sumber 4.4 berikut pada terminal.

```
sudo dpkg -i influxdb_0.13.0_amd64.deb
```

Kode Sumber 4.4: Command untuk installasi InfluxDB

InfluxDB akan digunakan sebagai penyimpanan data yang telah di kumpulkan oleh Collectd, untuk itu perlu dilakukan konfigurasi tambahan agar collectd pada setiap *docker host* dapat terhubung dengan influxdb yang terdapat pada *middleware* dengan menambahkan Kode Sumber 4.5 pada file `/etc/influxdb/influxdb.conf`.

```
[[collectd]]
  enabled = true
   bind-address = ":25826"
   database = "collectd"
   retention-policy = ""
   batch-size = 5000
   batch-pending = 10
   batch-timeout = "10s"
   read-buffer = 0
   typesdb = "/usr/share/collectd/types.db"
```

Kode Sumber 4.5: Konfigurasi tambahan InfluxDB

Setelah itu, buat file types.db folder /usr/share/collectd/, dengan isi file bisa didapatkan dari `https://github.com/collectd/collectd/blob/master/src/types.db`, lalu jalankan Kode Sumber 4.6 berikut untuk mengaktifkan konfigurasi yang baru saja dibuat.

```
sudo service influxdb restart
```

Kode Sumber 4.6: Command untuk menjalankan InfluxDB

4.4 Implementasi *Middleware*

Middleware merupakan komponen yang akan menerima perimantaan dari user, mengambil data dari basis data, menentukan *docker host* terbaik yang akan di pasangkan kontainer *docker* dan mengirimkan perintah untuk memasangkan kontainer *docker* pada *docker host* yang dipilih. Implementasi *middleware* akan terbagi menjadi implementasi basis data dan implementasi web service.

4.4.1 Implementasi *Web Service*

Pada implementasi *web service* dibutuhkan beberapa persiapan lingkungan implementasi yang perlu dilakukan, meliputi langkah-langkah berikut:

1. Instalasi *NodeJS* versi 6.10.2
2. Instalasi *Node* Packet Manager

Node JS akan berufungsi sebagai komponen dasar pembangun sistem, sedangkan *Node* Packet Manager (NPM) akan berfungsi untuk melakukan instalasi paket-paket *library* yang akan menjadi komponen pendukung dalam pembuatan sistem. Selain itu juga diperlukan instalasi paket-paket melalui NPM, meliputi:

1. Bluebird
2. Influx
3. Mathjs
4. Node-cmd
5. Express
6. Mysql

Paket paket tesebut akan di gunakan untuk mengambil data-data dari basis data mysql sebagai penyimpan data *docker host* yang tersedia untuk penyediaan kontainer, mengambil data-data dari influxDB sebagai data sumber daya setiap *docker host* yang ada dan juga mengimplementasikan pengambil keputusan dengan menggunakan algoritma AHP.

4.4.1.1 Rute *Web Service*

Middleware tidak memiliki antar muka grafis. Namun, diperlukan adanya rute-rute yang bisa diakses untuk melayani permintaan penyediaan kontainer dari user. Daftar rute yang disediakan oleh *middleware* tertera pada Tabel 4.1.

Tabel 4.1: Daftar Rute *Web Service*

HTTP Method	Rute	Parameter	Deskripsi
GET	/data	nama_kontainer	Berfungsi untuk memicu pengambilan keputusan *docker host* terbaik yang akan di pasangkan kontainer dengan algoritma *Analytic Hierarchy Process (AHP)*.

4.4.1.2 Pseduocode *Web Service*

Data *docker host* yang tersedia untuk dipasangkan kontainer akan didapatkan melalui basis data mysql yang ada. Dari data tersebut dapat di dapatkan total jumlah *docker host* yang tersedia. Jumlah tersebut akan digunakan sebagai parameter untuk mengambil data sumber daya *docker host* pada influxDB. setelah data sumber daya setiap *docker host* telah didapatkan, algoritma AHP akan menentukan *docker host* terbaik berdasarkan data-data tersebut. perintah untuk memasangkan kontainer pada *docker host* yang telah dipilih. Pada Kode Sumber 4.7 diperlihatkan bagaimana implementasinya dalam bentuk *pseudocode*.

Kode Sumber 4.7: Pseudocode Web Service

```
1   Get Number Of Server
2
3   for i=0; i<Number Of Server; i++
4     get Data Resource
5   Decision Making with AHP
6   Result = Server Hostname
7   command = ansible command + Result
```

4.4.2 Implementasi Basis Data

Berdasarkan hasil perancangan basis data pada bab terdapat 1 entitas yang diimplementasikan menjadi suatu tabel pada basis data MySQL. Detail implementasi entitas server tertera pada Kode Sumber 4.8.

```
CREATE TABLE server (
  id int PRIMARY KEY AUTO_INCREMENT,
  hostname VARCHAR(50)
);
```

Kode Sumber 4.8: *Query* untuk membuat tabel server

4.5 Implementasi *Pemasang Kontainer*

Pemasang kontainer yang digunakan pada sistem ini adalah perangkat lunak Ansible. Diperlukan beberapa tahap untuk dapat menggunakan Ansible, yaitu tahap pemasangan dan konfigurasi. Untuk melakukan pemasangan Ansible versi 3.0.0 pada sistem operasi Ubuntu jalankan kode sumber 4.9 pada terminal.

```
  sudo apt-add-repository ppa:ansible/ansible
    sudo apt-get update
    sudo apt-get install ansible
```

Kode Sumber 4.9: Perintah untuk installasi Ansible

Ansible menggunakan protokol SSH untuk dapat menjalankan perintah-perintah untuk membuat kontainer pada *docker host* yang dipilih, oleh karena itu ansible harus mengetahui hostname atau IP dari setiap *docker host* yang akan di pasangkan kontainer. hostname atau IP dari *docker host* tersebut akan di letakkan pada suatu file yang disebut inventory, secara *default* file inventory ansible terdapat pada /etc/ansible/host. Penulisan hostname atau IP *docker host* pada file inventory memiliki format seperti kode sumber 4.10

```
HOSTNAME ansible_user="user_pada_\textit{server}"
```

Kode Sumber 4.10: Format Inventory Ansible

Setelah itu, pada *server* host ansible, harus didaftarkan hostname dari setiap *docker host* yang akan di pasangkan kontainer, hal ini dapat dilakukan dengan mendaftarkan hostname dari *docker host* tersebut pada file /etc/hosts. Penulisan daftar hostname *docker host* pada file /etc/hosts memiliki format seperti kode sumber 4.11:

```
"IP \textit{server}"    "hostname \textit{server}"
contoh : 10.151.36.20    RAHWANA
```

Kode Sumber 4.11: Format file /etc/hosts

server host Ansible juga harus dapat melakukan ssh ke *docker host* yang ada tanpa menggunakan user dan password, hal ini dilakukan dengan cara menyalin ssh public key dari *server* host ansible ke setiap *docker host* yang akan dipasangkan kontainer. hal ini dapat dilakukan dengan perintah pada kode sumber **??** berikut:

```
ssh-keygen -t rsa
ssh-copy-id user@hostname
```

Kode Sumber 4.12: Perintah untuk menyalin ssh public key

Perintah pertama pada Kode Sumber 4.12 berguna untuk membuat key pada *server* host ansible, sedangkan perintah kedua

berguna untuk menyalin key pada *docker host* dengan hostname pada perintah tersebut. Setelah itu web service akan menjalankan perintah yang akan menjalankan ansible untuk memasangkan kontainer pada *server* yang diinginkan.

(Halaman ini sengaja dikosongkan)

BAB V

PENGUJIAN DAN EVALUASI

Pada bab ini akan dibahas uji coba dan evaluasi dari sistem yang telah dibuat. Sistem akan diuji coba fungsionalitas dan performanya dengan menjalankan skenario uji coba yang sudah ditentukan. Uji coba dilakukan untuk mengetahui hasi dari sistem ini sehingga dapat menjawab rumusan masalah pada tugas akhir ini.

5.1 Lingkungan Uji Coba

Uji coba sistem ini dilakukan dengan menggunakan 1 buah komputer penerima permintaan penyediaan kontainer, dan 4 komputer sebagai *docker host*.

1. **Komputer Penerima Permintaan Penyediaan Kontainer**

Tabel 5.1: Komputer Penerima Permintaan Penyediaan Kontainer

Perangkat Keras	Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
	RAM 8GB
	Hard disk 120GB
Perangkat Lunak	Linux Ubuntu 16.04 64 bit
	ExpressJS versi 4.15.2.
	MySQL versi 5.7.18.
	InfluxDB versi 0.13.0.
	Ansible Versi 3.0.0.
Konfigurasi Jaringan	IP address : 10.151.36.37
	Netmask : 255.255.255.0
	Gateway : 10.151.36.1
	Hostname : Laksmana

2. **Docker Host**
   (a) **Docker Host 1**

Tabel 5.2: Docker Host 1

Perangkat Keras	Processor Intel(R) Core(TM)2Duo CPU E7200 @ 2.53GHz
	RAM 2GB
	Hard disk 16GB
Perangkat Lunak	Linux Ubuntu 14.04.5 64 bit
	Collectd versi 5.6.2.35.
Konfigurasi Jaringan	IP address : 10.151.36.20
	Netmask : 255.255.255.0
	Gateway : 10.151.36.1
	Hostname : Rahwana

(b) **Docker Host 2**

Tabel 5.3: Docker Host 2

Perangkat Keras	Processor Intel(R) Core(TM)2Duo CPU E7200 @ 2.53GHz
	RAM 3GB
	Hard disk 50GB
Perangkat Lunak	Linux Ubuntu 14.04.5 64 bit
	Collectd versi 5.6.2.35.
Konfigurasi Jaringan	IP address : 10.151.36.21
	Netmask : 255.255.255.0
	Gateway : 10.151.36.1
	Hostname : Srikandi

(c) **Docker Host 3**

Tabel 5.4: Docker Host 3

Perangkat Keras	Processor Intel(R) Core(TM)2Duo CPU E7200 @ 2.53GHz
	RAM 2GB

	Hard disk 57GB
Perangkat Lunak	Linux Ubuntu 16.04.2 64 bit
	Collectd versi 5.6.2.35.
Konfigurasi Jaringan	IP address : 10.151.36.18
	Netmask : 255.255.255.0
	Gateway : 10.151.36.1
	Hostname : BALADEWA

(d) **Docker Host 4**

Tabel 5.5: Docker Host 4

Perangkat Keras	Processor Intel(R) Core(TM)2Duo CPU E7200 @ 2.53GHz
	RAM 2GB
	Hard disk 16GB
Perangkat Lunak	Linux Ubuntu 14.04.4 64 bit
	Collectd versi 5.6.2.35.
Konfigurasi Jaringan	IP address : 10.151.36.35
	Netmask : 255.255.255.0
	Gateway : 10.151.36.1
	Hostname : MEGANANDA

5.2 Skenario Uji Coba

Uji Coba ini dilakukan untuk menguji apakah fungsionalitas yang diidentifikasikan pada tahap kebutuhan benar-benar telah diimplementasikan dan bekerja seperti yang seharusnya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian respon sistem. Skenario pengujian dibedakan menjadi 2 bagian yaitu:

- **Uji Fungsionalitas** Pengujian ini didasarkan pada kebutuhan sistem yang telah didasarkan pada kebutuhan

sistem yang diidentifikasikan pada bab 3.

- **Uji Performa** Pengujian ini digunakan untuk mengukur efisiensi sistem dan performa sistem dalam menjalankankan fungsionalitas sistem.

5.2.1 Skenario Uji Fungsionalitas

Uji coba fungsionalitas dilakukan dengan cara menjalankan sistem yang telah dibuat, dan melakukan pengujian terhadap fitur yang telah dibuat. Uji coba fungsionalitas akan berfungsi untuk memastikan sistem sudah memenuhi kebutuhan yang tertera pada Bab 3, yaitu meliputi:

1. Pengujian User dapat mengirim Permintaan Penyediaan Kontainer.
2. Pengujian Sistem dapat melakukan penghitungan AHP, untuk menentukan *docker host*.
3. Pengujian *docker host* dapat menerima perintah penyediaan kontainer.
4. Pengujian *docker host* dapat Mengirim Data Resourcenya masing-masing.

5.2.1.1 Uji Coba User Mengirim Permintaan Penyediaan Kontainer

Uji coba ini dilakukan dengan mengakses sistem melalui rute /data dengan parameter image_name. Parameter image_name akan digunakan sebagai sisten untuk mengidentifikasi *image* docker yang akan dipasangkan pada *docker host*. Image *docker* yang digunakan pada ujicoba ini adalah *image* httpd yang akan menyediakan aplikasi *webserver* apache. Pengguna akan mengirim *http request* kepada web service yang telah disediakan pada Komputer Penerima Permintaan Penyediaan Kontainer. Daftar uji fungsionalitas Mengirim Permintaan Penyediaan Kontainer dijelaskan pada Tabel 5.6

Tabel 5.6: Skenario Uji Coba mengirim permintaan penyediaan kontainer

No	Routes	Uji Coba	Hasil Harapan
1	/data/httpd	Mengirim request menuju rute web service melalui browser.	*request* berhasil diterima oleh web service.

5.2.1.2 Uji Coba Sistem dapat Melakukan Penghitungan AHP

Uji coba ini dilakukan dengan mengakses sistem melalui rute /data dengan parameter image_name. Pengguna akan mengirim *http request* kepada web service yang telah disediakan pada Komputer Penerima Permintaan Penyediaan Kontainer dan sistem akan melakukan penghitungan AHP terhadap data sumber daya setiap *docker host* yang ada sebagai acuan.Data sumber daya yang dipakai sebagai acuan akan diambil dari basis data InfluxDB. Setelah penghitungan selesai diharapkan hostname dari *docker host* terbaik akan terpilih. Daftar uji fungsionalitas Sistem dapat melakukan Penghitungan AHP dijelaskan pada Tabel 5.7.

Tabel 5.7: Skenario Uji Coba Sistem dapat melakukan Penghitungan AHP

No	Routes	Uji Coba	Hasil Harapan
1	/data	Mengirim request menuju rute web service melalui browser.	web service berhasil melakukan AHP dan menghasilkan hostname dari *docker host* terpilih.

5.2.1.3 Uji Coba *Docker Host* dapat Menerima Perintah Penyediaan Kontainer

Uji coba ini dilakukan dengan mengakses sistem melalui rute /data dengan parameter image_name. Pengguna akan mengirim *http request* kepada web service yang telah disediakan pada Komputer Penerima Permintaan Penyediaan Kontainer dan sistem akan melakukan penghitungan AHP terhadap data sumber daya setiap *docker host* yang ada sebagai acuan. Setelah penghitungan selesai hostname dari *docker host* terbaik akan terpilih. Lalu sistem akan mengirimkan perintah penyediaan kontainer dengan ansbile ke *docker host* yang terpilih dan diharapkan *docker host* yang terpilih dapat menerima perintah tersebut dan menyediakan kontainer *docker* yang diinginkan. Pada Pengujian ini image yang digunakan untuk di pasangkan pada setiap *docker host* adalah image httpd, yaitu image untuk menyediakan *webserver* apache. Daftar uji fungsionalitas Sistem dapat menerima perintah penyediaan kontainer dijelaskan pada Tabel 5.8.

Tabel 5.8: Skenario Uji Coba *docker host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil dan Harapan
1	/data/httpd	*Docker Host* 1	Mengirim request menuju rute web service melalui browser.	*Docker Host* dapat menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* yang diinginkan.

Tabel 5.8: Skenario Uji Coba *docker host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil dan Harapan
2	/data/httpd	*Docker Host* 2	Mengirim request menuju rute web service melalui browser.	*Docker Host* dapat menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* yang diinginkan.
3	/data/httpd	*Docker Host* 3	Mengirim request menuju rute web service melalui browser.	*Docker Host* dapat menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* yang diinginkan.
4	/data/httpd	*Docker Host* 4	Mengirim request menuju rute web service melalui browser.	*Docker Host* dapat menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* yang diinginkan.

5.2.1.4 Uji Coba *Docker Host* dapat Mengirim Data Resourcenya Masing-Masing

Uji coba ini dilakukan dengan menanamkan pengumpul data pada setiap *docker host* dan mengarahkannya pada layanan penyimpanan data berbasis deret waktu yang terpasang pada *middleware*. Komputer *middleware* harus dapat mengakses *docker host* dan telah terpasang InfluxDB sebagai basis data berbasis deret waktu. Data yang dikumpulkan pada InfluxDB akan dijadikan acuan untukmelakukan pemngabilan keputusan menggunakan algoritma AHP. Daftar uji fungsionalitas *docker host* dapat Mengirim Data Resourcenya masing-masing dijelaskan pada Tabel 5.9

Tabel 5.9: Skenario Uji Coba *Docker Host* dapat Mengirim Data Resourcenya masing-masing

No	Docker Host	Uji Coba	Hasil Harapan
1	*Docker Host* 1	Mengirim data sumber daya ke InfluxDB pada *middleware* .	Data sumber daya berhasilkan dikirimkan dan tersimpan pada InfluxDB.
2	*Docker Host* 2	Mengirim data sumber daya ke InfluxDB pada *middleware* .	Data sumber daya berhasilkan dikirimkan dan tersimpan pada InfluxDB.

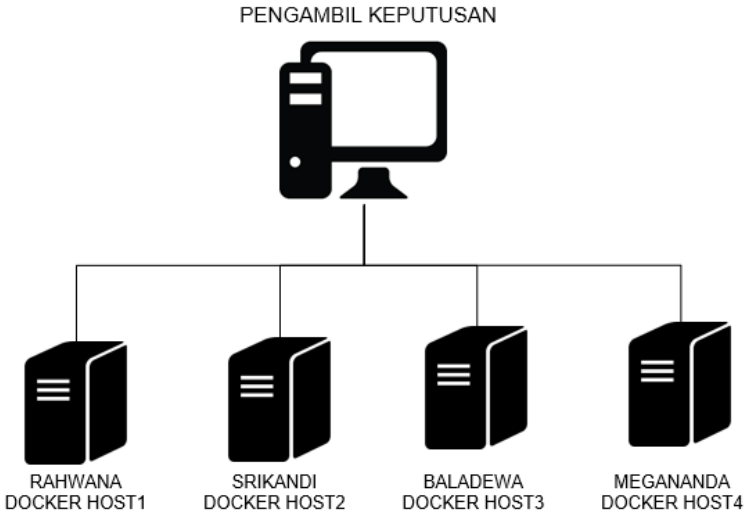
Tabel 5.9: Skenario uji Coba *Docker Host* dapat Mengirim Data Resource

No	Docker Host	Uji Coba	Hasil Harapan
3	*Docker Host* 3	Mengirim data sumber daya ke InfluxDB pada *middleware* .	Data sumber daya berhasilkan dikirimkan dan tersimpan pada InfluxDB.
4	*Docker Host* 4	Mengirim data sumber daya ke InfluxDB pada *middleware* .	Data sumber daya berhasilkan dikirimkan dan tersimpan pada InfluxDB.

5.2.2 Skenario Uji Performa

Sistem yang dibuat pada TA ini menggunakan algoritma *analytic hierarchy process* sebagai algoritma pengambilan keputusan untuk menentukan *docker host* terbaik untuk menyediakan kontainer *docker* yang diinginkan. AHP dipakai agar penempatan kontainer pada *docker host* dapat dilakukan secara efisien. Hal ini dilakukan dengan membandingkan ketersediaan sumberdaya pada masing-masing *docker host* dengan bobot priritas masing-masing kriteria sumber daya. Pengujian dilakukan dengan membandingkan hasil kerja sistem yang menggunakan AHP dengan sistem yang menggunakan algortima *round robin* untuk menentukan *docker host* yang akan dipilih untuk menyediakan kontainer. Setelah itu akan dilihat bagaimana performa sistem dengan algoritma AHP yang menggunakan multi kriteria dibandingkan dengan performa sistem yang menggunakan algortima *round robin* yang tidak menggunakan multi kriteria untuk menentukan *docker host* yang

akan dipilih. Selain itu pengujian juga akan menggunakan *image docker* variasi yaitu image httpd, nginx, moodle, dan mysql. Image yang bervariasi digunakan agar kebutuhan sumber daya masing-masing *docker host* yang dipakai berbeda-beda untuk setiap image *docker* yang akan dipasangkan pada setiap *docker host*. Selain itu, Uji Performa juga akan menilai kemampuan sistem dengan algoritma AHP bersarkan jumlah kontainer yang akan dipasangkan. Performa sistem akan di evaluasi secara bertahap dengan melihat pertumbuhan penggunan sumber daya pada masing-masing *docker host*. Uji coba performa akan menguji efisiensi sistem dalam menempatkan kontainer *docker* yang diinginkan oleh user pada *docker host* yang ada dan ketepatan sistem dalam menempatkan kontainer *docker* yang diinginkan. Arsitektur pengujian tertera pada gambar 5.1



Gambar 5.1: Arsitektur Pengujian Performa

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang sudah dijelaskan pada bab 5.2.

5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem.

5.3.1.1 Uji Coba User Mengirim Permintaan Penyediaan Kontainer

Uji coba ini dilakukan dengan mengakses sistem melalui rute dan parameter yang telah ditentukan pada Tabel 5.6. Pengguna akan mengirim *http request* kepada web service yang telah disediakan pada Komputer Penerima Permintaan Penyediaan Kontainer. Hasil uji coba seperti tertera pada tabel 5.10.

Tabel 5.10: Hasil Skenario Uji Coba mengirim permintaan penyediaan kontainer

No	Routes	Uji Coba	Hasil
1	/data/httpd	Mengirim request menuju rute web service melalui browser.	OK.

5.3.1.2 Uji Coba Sistem dapat Melakukan Penghitungan AHP

Uji coba ini dilakukan dengan mengakses sistem melalui rute yang telah ditentukan pada Tabel 5.7. Pengguna akan mengirim *http request* kepada web service yang telah disediakan pada

Komputer Penerima Permintaan Penyediaan Kontainer dan sistem akan melakukan penghitungan AHP terhadap data sumber daya setiap *docker host* yang ada sebagai acuan. Hasil uji coba seperti tertera pada Tabel 5.11 dan 5.12.

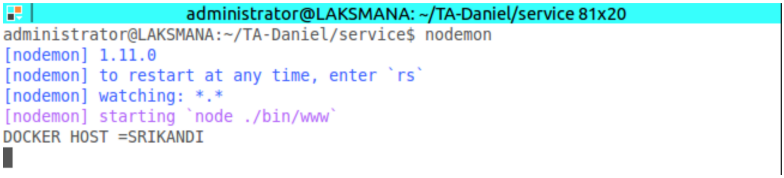
Tabel 5.11: Kondisi Awal Penggunaan Sumberdaya *Docker Host* sebelum uji coba dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	0.2%	797M/1.9G	11G/16G
2	SRIKANDI	0.3%	532M/2.9G	8.8G/50G
3	BALADEWA	0.1%	354M1.9G/	37G/57G
4	MEGANANDA	0.1%	555M/1.9G	10G/16G

Tabel 5.12: Hasil Skenario Uji Coba Sistem dapat melakukan Penghitungan AHP

No	Docker Host	Uji Coba	Hasil
1	/data/httpd	Mengirim request menuju rute web service melalui browser.	web service berhasil melakukan AHP dan menghasilkan hostname dari *docker host* terpilih, yaitu dalam uji coba ini menghasilkan *hostname* SRIKANDI.

Pada Uji Coba berhasil didapatkan hostname dari *Docker Host* terbaik, yaitu SRIKANDI seperti yang ditunjukkan pada Gambar 5.2, dikarenakan SRIKANDI memiliki ketersediaan sumber daya yang lebih baik dibanding *Docker Host* lain berdasarkan penghitungan dengan menggunakan algortima AHP.



Gambar 5.2: Gambar Hasil Uji Sistem dapat melakukan Penghitungan AHP

5.3.1.3 Uji Coba *Docker Host* dapat Menerima Perintah Penyediaan Kontainer

Uji coba ini dilakukan dengan mengakses sistem melalui rute, parameter yang telah ditentukan pada Tabel 5.8. Pengujian dilakukan image httpd yang akan diujikan pada setiap *docker host*. Hasil uji coba ditunjukkan pada Tabel 5.13

Tabel 5.13: Hasil Skenario Uji Coba *Docker Host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil
1	/data/httpd	*Docker Host* 1	Mengirim request penyediaan *docker* dengan image httpd menuju rute web service melalui browser.	*Docker Host* berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* dengan image httpd.

Tabel 5.13: Hasil Skenario Uji Coba *Docker Host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil
2	/data/httpd	*Docker Host* 2	Mengirim request penyediaan *docker* dengan image httpd menuju rute web service melalui browser.	*Docker Host* berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* dengan image httpd.
3	/data/httpd	*Docker Host* 3	Mengirim request penyediaan *docker* dengan image httpd menuju rute web service melalui browser.	*Docker Host* berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* dengan image httpd.

Tabel 5.13: Hasil Skenario Uji Coba *Docker Host* dapat menerima perintah penyediaan kontainer

No	Route	Docker Host	Uji Coba	Hasil
4	/data/httpd	*Docker Host* 4	Mengirim request penyediaan *docker* dengan image httpd menuju rute web service melalui browser.	*Docker Host* berhasil menerima perintah penyediaan kontainer dan menyediakan kontainer *docker* dengan image httpd.

Pada Gambar 5.3 dan Gambar 5.4 ditunjukkan bagaimana kondisi saat sistem menghasilkan perintah untuk menyediakan *docker* dan saat *docker* sudah terpasang pada *docker host*

Gambar 5.3: Web Service Mengrimkan perintah penyediaan kontainer

Gambar 5.4: *Docker Host* berhasil membuat kontainer *docker*

5.3.1.4 Uji Coba *Docker Host* dapat Mengirim Data Resourcenya Masing-Masing

Dilakukan pengujian pada setiap *docker host*. Setiap *docker host* akan mengirimkan data ketersediaan sumber daya CPU, RAM dan File Storage nya masing-masing ke InfluxDB yang terdapat pada *middleware*. Hasil ditunjukkan pada Tabel 5.14

Tabel 5.14: Hasil Skenario Uji Coba *Docker Host* dapat Mengirim Data Resourcenya Masing-Masing

No	Docker Host	Uji Coba	Hasil
1	*Docker Host* 1	Mengirim data sumber daya (RAM, CPU dan *File Storage*) ke InfluxDB pada *middleware* .	Data sumber daya CPU, RAM dan File Storage berhasilkan dikirimkan dan tersimpan pada InfluxDB.

Tabel 5.14: Hasil Skenario uji Coba *Docker Host* dapat Mengirim Data Resource

No	Docker Host	Uji Coba	Hasil
2	*Docker Host* 2	Mengirim data sumber daya (RAM, CPU dan *File Storage*) ke InfluxDB pada *middleware* .	Data sumber daya CPU, RAM dan File Storage berhasilkan dikirimkan dan tersimpan pada InfluxDB.
3	*Docker Host* 3	Mengirim data sumber daya (RAM, CPU dan *File Storage*) ke InfluxDB pada *middleware* .	Data sumber daya CPU, RAM dan File Storage berhasilkan dikirimkan dan tersimpan pada InfluxDB.
4	*Docker Host* 4	Mengirim data sumber daya (RAM, CPU dan *File Storage*) ke InfluxDB pada *middleware* .	Data sumber daya CPU, RAM dan File Storage berhasilkan dikirimkan dan tersimpan pada InfluxDB.

Sesuai dengan Gambar 5.5,5.6 dan 5.7, ,semua *docker host* berhasil mengirimkan data sumber daya CPU, RAM dan File Storage nya masing-masing ke InfluxDB.

- Data CPU Setiap Docker Host pada InfluxDB

```
key
cpu_value,host=BALADEWA,instance=0,type=percent,type_instance=idle
cpu_value,host=BALADEWA,instance=1,type=percent,type_instance=idle
cpu_value,host=MEGANANDA,instance=0,type=percent,type_instance=idle
cpu_value,host=RAHWANA,instance=0,type=percent,type_instance=idle
cpu_value,host=RAHWANA,instance=1,type=percent,type_instance=idle
cpu_value,host=SRIKANDI,instance=0,type=percent,type_instance=idle
cpu_value,host=SRIKANDI,instance=1,type=percent,type_instance=idle
```

Gambar 5.5: Data CPU Setiap Docker Host pada InfluxDB

- Data RAM Setiap Docker Host pada InfluxDB

```
memory_value,host=BALADEWA,type=memory,type_instance=free
memory_value,host=BALADEWA,type=percent,type_instance=free
memory_value,host=MEGANANDA,type=memory,type_instance=free
memory_value,host=MEGANANDA,type=percent,type_instance=free
memory_value,host=RAHWANA,type=memory,type_instance=free
memory_value,host=RAHWANA,type=percent,type_instance=free
memory_value,host=SRIKANDI,type=memory,type_instance=free
memory_value,host=SRIKANDI,type=percent,type_instance=free
```

Gambar 5.6: Data RAM Setiap Docker Host pada InfluxDB

- Data Penyimpanan File Setiap Docker Host pada InfluxDB

```
df_value,host=BALADEWA,instance=root,type=df_complex,type_instance=free
df_value,host=BALADEWA,instance=root,type=percent_bytes,type_instance=free
df_value,host=MEGANANDA,instance=root,type=df_complex,type_instance=free
df_value,host=MEGANANDA,instance=root,type=percent_bytes,type_instance=free
df_value,host=RAHWANA,instance=root,type=df_complex,type_instance=free
df_value,host=RAHWANA,instance=root,type=percent_bytes,type_instance=free
df_value,host=SRIKANDI,instance=root,type=df_complex,type_instance=free
df_value,host=SRIKANDI,instance=root,type=percent_bytes,type_instance=free
```

Gambar 5.7: Data Penyimpanan File Setiap Docker Host pada InfluxDB

5.3.2 Uji Performa

Seperti yang dijelaskan pada bab 5.2 pengujian performa akan dilakukan pada 4 *docker host* yang tersedia dengan menggunakan sistem yang ada, namun dengan 2 algoritma yang berbeda yaitu menggunakan algoritma pengambilan keputusan AHP dan menggunakan algoritma pembagian kerja dasar, *round robin*[7]. Selain itu pengujian juga akan menggunakan beberapa

image *docker* yang akan di pasangkan pada setiap *docker host* yang tersedia. Pengujian akan dijalankan dengan mengirimkan permintaan penyediaan kontainer pada sistem dengan jumlah permintaan penyediaan kontainer yang beragam.

5.3.2.1 Pengujian Dengan Algoritma AHP

Kondisi awal ketersediaan sumberdaya *docker host* sebelum uji coba sistem dengan image *docker* httpd dan nginx menggunakan AHP dijalankan ditunjukkan pada Tabel 5.15.

Tabel 5.15: Kondisi Awal Ketersediaan Sumberdaya *Docker Host* sebelum uji coba dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	98%	1.3G/1.9G	5G/16G
2	SRIKANDI	93%	2.3G/2.9G	41.2G/50G
3	BALADEWA	99%	1G/1.9G	20G/57G
4	MEGANANDA	91%	1.3G/1.9G	6G/16G

Hasil pengujian sistem dengan algoritma AHP menggunakan *image docker* httpd dan nginx ditunjukkan pada Tabel 5.16.

Tabel 5.16: Hasil Uji Coba Performa sistem dengan *Image Docker* httpd dan nginx menggunakan AHP

No	Jumlah Kontainer	Kontainer Terpasang pada *Docker Host*			
		Rahwana	Srikandi	Baladewa	Megananda
1	254	39	135	27	53
2	254	40	131	32	51
3	254	37	147	20	50
4	254	41	143	23	47
5	254	49	132	24	49

Kondisi akhir ketersediaan sumber daya *docker host* pada

pengujian sistem dengan algoritma AHP menggunakan *image docker* httpd dan nginx ditunjukkan pada Tabel 5.17.

Tabel 5.17: Rata-rata Kondisi Akhir Ketersediaan Sumberdaya *Docker Host* Setelah Uji Coba Dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	90%	628M/1.9G	5G/16G
2	SRIKANDI	85%	708M/2.9G	41.2G/50G
3	BALADEWA	92%	832M/1.9G	20G/57G
4	MEGANANDA	90%	654M/1.9G	6G/16G

5.3.2.2 Pengujian Dengan Algoritma *Round Robin*

Kondisi awal ketersediaan sumberdaya *docker host* sebelum uji coba sistem dengan *image docker* httpd dan nginx menggunakan *round robin* dijalankan ditunjukkan pada Tabel 5.18.

Tabel 5.18: Kondisi Awal Ketersediaan Sumberdaya *Docker Host* Sebelum Uji Coba Dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	98%	1.2G/1.9G	5G/16G
2	SRIKANDI	93%	2.3G/2.9G	41.2G/50G
3	BALADEWA	99%	1G/1.9G	20G/57G
4	MEGANANDA	91%	1.4G/1.9G	6G/16G

Hasil dari pengujian sistem dengan algoritma *round robin* ditunjukkan pada Tabel 5.19.

Tabel 5.19: Hasil Uji Coba Performa Sistem dengan *Image Docker* httpd dan nginx Menggunakan *Round Robin*

No	Jumlah Kontainer	Kontainer Terpasang pada Docker Host			
		Rahwana	Srikandi	Baladewa	Megananda

1	254	63	63	63	63
2	254	63	63	63	63
3	254	63	63	63	63
4	254	63	63	63	63
5	254	63	63	63	63

Kondisi akhir ketersediaan sumberdaya *docker host* pada pengujian sistem dengan algoritma *round robin* menggunakan *image docker* httpd dan nginx ditunjukkan pada Tabel 5.20.

Tabel 5.20: Kondisi Akhir Ketersediaan Sumberdaya *Docker Host* Setelah Uji Coba Dijalankan

No	Docker Host	CPU	RAM	Storage
1	RAHWANA	92%	161M/1.9G	5G/16G
2	SRIKANDI	93%	1.3G/2.9G	41.1G/50G
3	BALADEWA	92%	95M/1.9G	20G/57G
4	MEGANANDA	91%	335M/1.9G	6G/16G

5.3.2.3 Pengujian Performa Sistem dengan Algoritma AHP Berdasarkan Jumlah Kontainer yang Akan Dipasangkan

Kondisi awal ketersediaan sumberdaya *docker host* sebelum uji coba ditunjukkan pada Tabel 5.21.

Tabel 5.21: Kondisi Awal Ketersediaan Sumberdaya *Docker Host* Sebelum Uji Coba Dijalankan

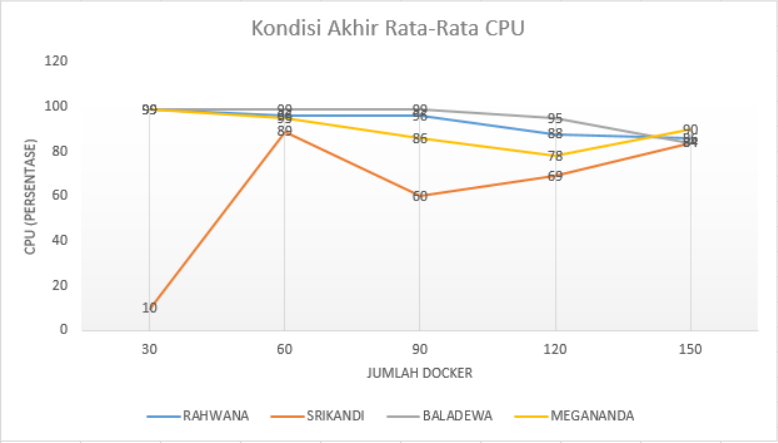
No	Docker Host	CPU	RAM	Storage
1	RAHWANA	99%	1.3G/1.9G	3.8G/16G
2	SRIKANDI	99%	2.3G/2.9G	37G/50G
3	BALADEWA	99%	1G/1.9G	16G/57G
4	MEGANANDA	99%	1.4G/1.9G	3.8G/16G

Hasil pengujian sistem dengan algoritma AHP menggunakan *docker image* httpd, nginx, moodle, dan mysql ditunjukkan pada Table 5.22.

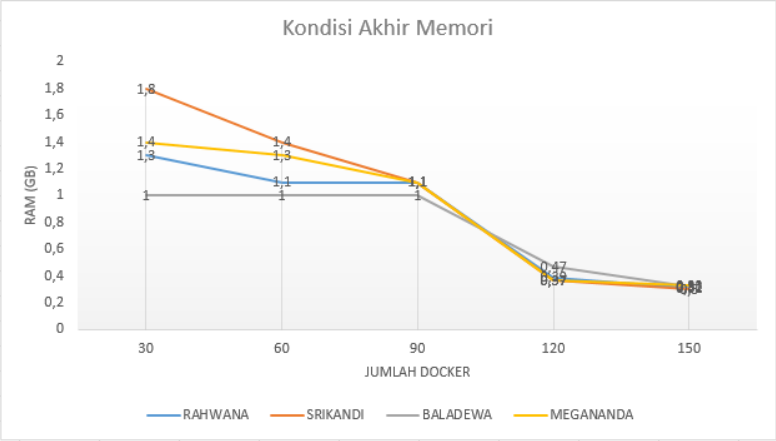
Tabel 5.22: Hasil Uji Coba Performa Sistem dengan *Docker Image* httpd dan nginx Menggunakan AHP

No	Jumlah Kontainer	Kontainer Terpasang pada Docker Host			
		Rahwana	Srikandi	Baladewa	Megananda
1	30	0	30	0	0
2	60	3	55	0	2
3	100	5	73	0	12
4	120	11	88	5	16
5	150	34	46	33	37

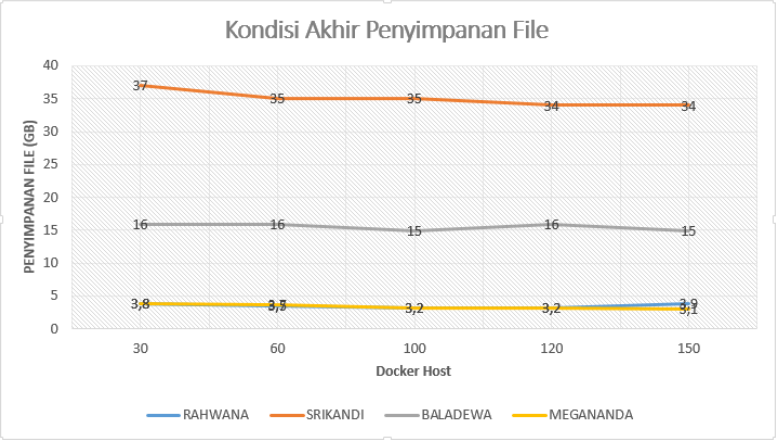
Kondisi akhir ketersediaan sumberdaya *docker host* setelah uji coba dijalankan ditunjukkan pada Gambar 5.8 untuk ketersediaan CPU, Gambar 5.9 untuk ketersediaan Memori dan Gambar 5.10 untuk ketersediaan penyimpaan berkas.



Gambar 5.8: Grafik Kondisi Akhir Ketersediaan Rata-Rata CPU

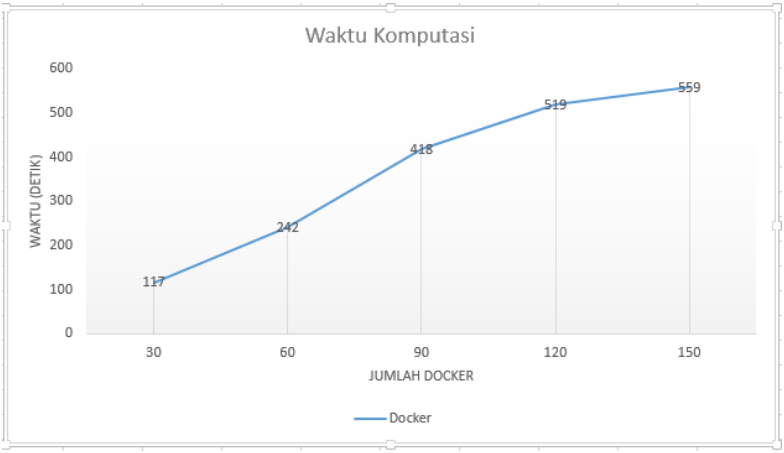


Gambar 5.9: Grafik Kondisi Akhir Ketersediaan Memori



Gambar 5.10: Grafik Kondisi Akhir Ketersedian Penyimpaanan FIle

Waktu Pendistribusian Docker oleh Sistem Berdasarkan Jumlah Kontainer ditunjukkan pada Gambar 5.11



Gambar 5.11: Grafik Waktu Pendistribusian Docker oleh Sistem Berdasarkan Jumlah Kontainer

Dari data hasil uji coba yang didapat, dapat dilihat bahwa algoritma AHP dapat membagikan kontainer ke *docker host* yang ada dengan lebih efisien. Dibandingkan dengan algoritma *round robin* yang menyebarkan kontainer merata ke setiap *docker host*, sistem dengan AHP menghasilkan kondisi ketersediaan sumber daya yang lebih merata pada akhir uji coba. Selain itu pendistribusian pada setiap *docker host* juga memiliki ketepatan yang baik, hal ini terlihat pada saat akan mendistribusikan 30 kontainer, sistem dengan AHP mendistribusikan seluruh kontainer tersebut pada *docker host* SRIKANDI, hal ini disebabkan SRIKANDI memiliki ketersediaan yang jauh lebih dominan dibanding *docker host* yang lain. Namun, kondisi dimana *docker image* yang bervariasi menimbulkan hasil akhir dari uji coba tidak linear, hal ini disebabkan variasi dari *docker image* akan membuat kebutuhan sumber daya yang dibutuhkan dari setiap *docker host* juga bervariasi.

BAB VI

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Sistem dapat mendistribusikan 100% penyediaan kontainer ke *docker host* yang ada dengan multi kriteria dengan cara mendistirbusikan perintah penyediaan kontainer *docker* melalui protokol ssh.
2. Sistem dapat menentukan penyedia kontainer dengan menggunakan algoritma AHP dengan kriteria-kriteria seperti penggunaan CPU, RAM , dan Penyimpanan File pada masing-masing *docker host* yang ada.
3. Dengan menggunakan AHP, dibandingkan dengan metode *round robin*, sistem dapat menentukan *docker host* yang akan menyediakan kontainer docker dengan efisien berdasarkan penggunaan CPU, RAM , dan File Storage pada masing-masing *docker host* yang ada, dimana dengan menggunakan AHP ketersediaan akhir memori dari setiap *docker host* lebih merata dengan rentang 628MB sampai 832MB, sedangkan dengan *round robin* ketersediaan akhir memori sangat tidak merata, dengan rentang yang cukup jauh dimulai 95MB sampai 1.3GB.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Sistem dapat dikembangkan dengan menambahkan kriteria-kriteria yang sesuai dengan lingkungan sistem yang ada, seperti jarak antara *docker host* dan *server* middleware atau kecepatan bandwith dari setiap *docker host* merupakan kriteria yang lebih baik untuk sistem yang memasangkan kontainer yang tidak memiliki perkembangan penggunaan pada aplikasi yang terdapat pada kontainer. Sedangkan sumber daya seperti file storage dapat digunakan untuk sistem yang mengalami perkembangan penggunaan pada aplikasi yang terdapat pada kontainernya.
- AHP merupakan algoritma MCDM yang paling umum digunakan dikarenakan proses yang tidak rumit dan memiliki unsur objektif dan subjektif dalam pengambilan keputusannya. Untuk pengembangan kedepannya sistem dapat diimplementasikan dengan menggunakan algortima MCDM lain untuk meningkatkan performa sistem, seperi algoritma Fuzzy AHP.

DAFTAR PUSTAKA

[1] F. Mohammad, V. Yadav, and others, "Automatic decision making for multi-criteria load balancing in cloud environment using AHP," in *Computing, Communication & Automation (ICCCA), 2015 International Conference on*, pp. 569–576, IEEE, 2015.

[2] S. Tilkov and S. Vinoski, "Node. js: Using JavaScript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[3] L. Hochstein, *Ansible: Up and Running*. " O'Reilly Media, Inc.", 2014.

[4] J. S. Ward and A. Barker, "Self managing monitoring for highly elastic large scale cloud deployments," in *Proceedings of the sixth international workshop on Data intensive distributed computing*, pp. 3–10, ACM, 2014.

[5] B. Leighton, S. J. Cox, N. J. Car, M. P. Stenson, J. Vleeshouwer, and J. Hodge, "A best of both worlds approach to complex, efficient, time series data delivery," in *International Symposium on Environmental Software Systems*, pp. 371–379, Springer, 2015.

[6] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[7] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on networking*, vol. 4, no. 3, pp. 375–385, 1996.

(Halaman ini sengaja dikosongkan)

LAMPIRAN A

FILE KONFIGURASI

1.1 File Konfigurasi Collectd

Berikut File Konfigurasi Keseluruhan untuk perangkat lunak Collectd.

```
FQDNLookup true
LoadPlugin syslog

<Plugin syslog>
  LogLevel info
</Plugin>

LoadPlugin cpu
LoadPlugin df
LoadPlugin memory
LoadPlugin network

<Plugin cpu>
  ReportByCpu true
  ReportByState true
  ValuesPercentage true
</Plugin>


<Plugin memory>
  ValuesAbsolute true
  ValuesPercentage true
</Plugin>

<Plugin df>
  Device "/dev/sda6"
  MountPoint "/"
  FSType "ext4"

  IgnoreSelected false

  ReportInodes false

  ValuesAbsolute true
  ValuesPercentage true
</Plugin>

<Plugin network>
  Server "10.151.36.37" "25826"
</Plugin>
```

```
<Include "/etc/collectd/collectd.conf.d">
  Filter "*.conf"
</Include>
```

Kode Sumber 1.1: Kode sumber Model Auth

1.2 File Konfigurasi InfluxDB

Berikut File Konfigurasi Keseluruhan untuk perangkat lunak InfluxDB.

```
reporting-disabled = false

[meta]
  # Where the metadata/raft database is stored
  dir = "/var/lib/influxdb/meta"

  retention-autocreate = true

  # If log messages are printed for the meta service
  logging-enabled = true
  pprof-enabled = false

  # The default duration for leases.
  lease-duration = "1m0s"

[data]
  # Controls if this node holds time series data shards in the
      cluster
  enabled = true

  dir = "/var/lib/influxdb/data"

  # These are the WAL settings for the storage engine >= 0.9.3
  wal-dir = "/var/lib/influxdb/wal"
  wal-logging-enabled = true
  data-logging-enabled = true

###
### [cluster]
###
### Controls non-Raft cluster behavior, which generally includes
     how data is
```

```
### shared across shards.
###

[cluster]
  shard-writer-timeout = "5s" # The time within which a remote
       shard must respond to a write request.
  write-timeout = "10s" # The time within which a write request
       must complete on the cluster.
  max-concurrent-queries = 0 # The maximum number of concurrent
       queries that can run. 0 to disable.
  query-timeout = "0s" # The time within a query must complete
       before being killed automatically. 0s to disable.
  max-select-point = 0 # The maximum number of points to scan in a
       query. 0 to disable.
  max-select-series = 0 # The maximum number of series to select in
        a query. 0 to disable.
  max-select-buckets = 0 # The maximum number of buckets to select
       in an aggregate query. 0 to disable.

###
### [retention]
###
### Controls the enforcement of retention policies for evicting old
      data.
###

[retention]
  enabled = true
  check-interval = "30m"

###
### [shard-precreation]
###
### Controls the precreation of shards, so they are available
     before data arrives.
### Only shards that, after creation, will have both a start- and
     end-time in the
### future, will ever be created. Shards are never precreated that
     would be wholly
### or partially in the past.

[shard-precreation]
  enabled = true
  check-interval = "10m"
  advance-period = "30m"

###
```

```
### Controls the system self-monitoring, statistics and diagnostics
    .
###
### The internal database for monitoring data is created
    automatically if
### if it does not already exist. The target retention within this
    database
### is called 'monitor' and is also created with a retention period
     of 7 days
### and a replication factor of 1, if it does not exist. In all
    cases the
### this retention policy is configured as the default for the
    database.

[monitor]
  store-enabled = true # Whether to record statistics internally.
  store-database = "_internal" # The destination database for
      recorded statistics
  store-interval = "10s" # The interval at which to record
      statistics

###
### [admin]
###
### Controls the availability of the built-in, web-based admin
    interface. If HTTPS is
### enabled for the admin interface, HTTPS must also be enabled on
    the [http] service.
###

[admin]
  enabled = true
  bind-address = ":8083"
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"

###
### [http]
###
### Controls how the HTTP endpoints are configured. These are the
    primary
### mechanism for getting data into and out of InfluxDB.
###

[http]
  enabled = true
  bind-address = ":8086"
```

```
  auth-enabled = false
  log-enabled = true
  write-tracing = false
  pprof-enabled = false
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"
  max-row-limit = 10000

###
### [[graphite]]
###
### Controls one or many listeners for Graphite data.
###

[[graphite]]
  enabled = false


###
### [collectd]
###
### Controls one or many listeners for collectd data.
###

[[collectd]]
  enabled = true
   bind-address = ":25826"
   database = "collectd"
   retention-policy = ""
  # These next lines control how batching works. You should have
       this enabled
  # otherwise you could get dropped metrics or poor performance.
       Batching
  # will buffer points in memory if you have many coming in.

   batch-size = 5000 # will flush if this many points get buffered
   batch-pending = 10 # number of batches that may be pending in
        memory
   batch-timeout = "10s" # will flush at least this often even if
        we haven't hit buffer limit
   read-buffer = 0 # UDP Read buffer size, 0 means OS default. UDP
        listener will fail if set above OS max.
   typesdb = "/usr/share/collectd/types.db"

###
### [opentsdb]
###
```

```
### Controls one or many listeners for OpenTSDB data.
###

[[opentsdb]]
  enabled = false


###
### [[udp]]
###
### Controls the listeners for InfluxDB line protocol data via UDP.
###

[[udp]]
  enabled = false


###
### [continuous_queries]
###
### Controls how continuous queries are run within InfluxDB.
###

[continuous_queries]
  log-enabled = true
  enabled = true
  # run-interval = "1s" # interval for how often continuous queries
        will be checked if they need to run
```

Kode Sumber 1.2: Kode sumber Model Auth

1.3 File Inventory Ansible

Berikut File Inventory dari perangkat lunak Ansible.

```
[worker]
RAHWANA ansible_user=administrator
SRIKANDI ansible_user=administrator
BALADEWA ansible_user=administrator
MEGANANDA ansible_user=administrator
```

Kode Sumber 1.3: Kode sumber Model Auth

LAMPIRAN B

KODE SUMBER

2.1 Web Service

Berikut File Inventory dari perangkat lunak Ansible.

```
var express = require('express');
var mysql = require('mysql');
var Influx = require('influx');
var router = express.Router();
var conn = require('../db-mysql.js');
var Servers=require('../models/Servers');
var math = require('mathjs');
var cmd = require('node-cmd');
var Promise = require('bluebird');

var flag = 1;
//apache and nginx
var port = 8080;

var influx = new Influx.InfluxDB({
  host: '10.151.36.37',
  database: 'collectd'
})

function indexOfMax(arr) {
  if (arr.length === 0) {
      return -1;
  }

  var max = arr[0];
  var maxIndex = 0;

  for (var i = 1; i < arr.length; i++) {
      if (arr[i] > max) {
          maxIndex = i;
          max = arr[i];
      }
  }

  return maxIndex;
}


router.get('/', function(req,res){
  var query="SELECT * FROM servers";
  var resources=[];
  var counter=0
```

```
var df=0
var cpu=0
conn.query(query, function(err, rows, fields) {
  if(err) {
    return res.json({"Error" : true, err});
  } else {
      servers=rows.length;
      for (var i = 0; i < servers; i++) {
        output= {
          'hostname'   :'',
          'memory' : '',
          'cpu'     : '',
          'df'  : ''
        }
        resources.push(output);
        Promise.all([
         influx.query(`select last("value") from memory_value
              where type='percent' and type_instance='free' and
              time > now() - 24h and host=${Influx.escape.
              stringLit(rows[i].hostname)} group by host`),
         influx.query(`select mean("value") from cpu_value where
              type='percent' and type_instance='idle' and time >
              now() - 24h and host=${Influx.escape.stringLit(rows[
              i].hostname)} group by host`),
         influx.query(`select last("value") from df_value where
              type='percent_bytes' and type_instance='free' and
              time > now() - 24h and host=${Influx.escape.
              stringLit(rows[i].hostname)} group by host`)
          ]).spread(function(query1,query2,query3){
            // console.log(query3)
            resources[counter].hostname=query1[0].host
            resources[counter].memory=query1[0].mean
            resources[counter].cpu=query2[0].mean
            resources[counter].df=query3[0].last
            counter++
            if(counter==servers){
              //AHP
              //Comparison Matrix
              // [     CPU  MEM   DF ]
              // [CPU   1   0.25  0.5]
              // [MEM   4    1     5 ]
              // [DF    2   0.2    1 ]
              var CM = math.matrix([[1, 0.25, 0.5,0,0], [4, 1,
                   5,0,0], [2, 0.2, 1,0,0]]);
              //get 3rd root of Comparison Matrix
              CM.subset(math.index(0, 3),math.pow((CM.subset(math
                   .index(0, 0))*CM.subset(math.index(0, 1))*CM.
```

```
     subset(math.index(0, 2))), 1/3));
CM.subset(math.index(1, 3),math.pow((CM.subset(math
     .index(1, 0))*CM.subset(math.index(1, 1))*CM.
     subset(math.index(1, 2))), 1/3));
CM.subset(math.index(2, 3),math.pow((CM.subset(math
     .index(2, 0))*CM.subset(math.index(2, 1))*CM.
     subset(math.index(2, 2))), 1/3));
//get priority vector of Comparison Matrix
var sum = CM.subset(math.index(0, 3))+CM.subset(
     math.index(1, 3))+CM.subset(math.index(2, 3));
CM.subset(math.index(0, 4),CM.subset(math.index(0,
     3)) / sum)
CM.subset(math.index(1, 4),CM.subset(math.index(1,
     3)) / sum)
CM.subset(math.index(2, 4),CM.subset(math.index(2,
     3)) / sum)

//getting rate of each server
var range = servers + 2;
var mem = math.ones(servers,range)
var cpu = math.ones(servers,range)
var df = math.ones(servers,range)
for(i=0;i < servers; i++){
  for (j = 0; j < servers; j++) {
      mem.subset(math.index(i, j), resources[i].
           memory/resources[j].memory)
      cpu.subset(math.index(i, j), resources[i].cpu
           /resources[j].cpu)
      df.subset(math.index(i, j), resources[i].df/
           resources[j].df)
    }
}

var priority = []
for (var i = 0; i < servers; i++) {
   priority.push({
      memory:'',
      cpu:'',
      df:''
   });
 }

//get 3rd root of product(rop) and priority of MEM
var rop_mem=[];
var ropmsum=0;
for(i=0;i < servers; i++){
  var ropmval=1;
```

```
                for (j = 0; j < servers; j++) {
                    ropmval = ropmval * mem.subset(math.index(i,
                        j))
                    if(j==servers-1){
                      // console.log(ropmval)
                      rop_mem[i]=math.pow(ropmval,1/3);
                      ropmsum = ropmsum + rop_mem[i];
                      }
                }
                if(i==servers-1)
                var pmval=1;
                for(k=0;k < rop_mem.length; k++){
                  pmval = rop_mem[k]/ropmsum;
                  priority[k].memory=pmval;
                }
              }

              //get 3rd root of product(rop) and priority of DF
              var rop_df=[];
              var ropdsum=0;
              for(i=0;i < servers; i++){
                var ropdval=1;
                for (j = 0; j < servers; j++) {
                    ropdval = ropdval * df.subset(math.index(i, j
                        ))
                    if(j==servers-1){
                      // console.log(ropmval)
                      rop_df[i]=math.pow(ropdval,1/3);
                      ropdsum = ropdsum + rop_df[i];
                      }
                }
                if(i==servers-1)
                var pdval=1;
                for(k=0;k < rop_df.length; k++){
                  pdval = rop_df[k]/ropdsum;
                  priority[k].df=pdval;
                }
              }

              //get 3rd root of product(rop) and priority of CPU
              var rop_cpu=[];
              var ropcsum=0;
              for(i=0;i < servers; i++){
                var ropcval=1;
                for (j = 0; j < servers; j++) {
                    ropcval = ropcval * cpu.subset(math.index(i,
                        j))
```

```
                    if(j==servers-1){
                      // console.log(ropmval)
                      rop_cpu[i]=math.pow(ropcval,1/3);
                      ropcsum = ropcsum + rop_cpu[i];
                      }
                  }
                  if(i==servers-1)
                  var pcval=1;
                  for(k=0;k < rop_cpu.length; k++){
                    pcval = rop_cpu[k]/ropcsum;
                    priority[k].cpu=pcval;
                  }
                }
                //getting score => SIgma(priority per criteria *
                     weifgth of node for each criteria)
                var criteria = 3
                var score=[]
                for (var i = 0; i < servers; i++) {
                    score[i] = CM.subset(math.index(0, 4))*priority
                         [i].cpu + CM.subset(math.index(1, 4))*
                         priority[i].memory + CM.subset(math.index
                         (2, 4))*priority[i].df
                }
                var decision = indexOfMax(score);

                command = 'ansible-playbook /home/administrator/TA-
                     Daniel/ansible-test/playbooks/apache.yml --
                     extra-vars "port='+port+' host='+resources[
                     decision].hostname+' name=container'+flag+'"';
                flag ++;
                port ++;
                cmd.get(
                    command,
                    function(err, data, stderr){
                        res.json(data)
                    }
                );
              }
            });
        }
    }
  });
})
module.exports=router;
```

Kode Sumber 2.1: Kode sumber Model Auth

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Daniel Fablius, lahir pada tanggal 21 Februari 1995 di Jakarta. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain futsal dan DOTA. Pernah menjadi asisten praktikum mata kuliah Sistem Operasi dan Jaringan Komputer. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Pengembangan Sumber Daya Manusia Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-2, serta Kepala Divisi Kaderisasi Departemen Kaderisasi dan Pemetaan Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-3.