



ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA RANCANG BANGUN SISTEM MONITORING PERANGKAT JARINGAN DI ITS

AFIF RIDHO KAMAL PUTRA
NRP 05111440000173

Dosen Pembimbing I
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D

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

JURUSAN DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

**IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA RANCANG
BANGUN SISTEM MONITORING PERANGKAT JARINGAN DI
ITS**

AFIF RIDHO KAMAL PUTRA
NRP 05111440000173

Dosen Pembimbing I
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D

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

JURUSAN DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

**IMPLEMENTATION OF PUBLISH/SUBSCRIBE ON DESIGN
OF NETWORK MONITORING DEVICE SYSTEM IN ITS**

AFIF RIDHO KAMAL PUTRA
NRP 05111440000173

Supervisor I
Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D

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

Department of INFORMATICS
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN
IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA
RANCANG BANGUN SISTEM MONITORING
PERANGKAT JARINGAN DI ITS

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Jurusan Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :

AFIF RIDHO KAMAL PUTRA
NRP: 05111440000173

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D
NIP: 198407082010122004 (Pembimbing 1)

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

SURABAYA
Juni 2018

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI *PUBLISH/SUBSCRIBE* PADA RANCANG BANGUN SISTEM MONITORING PERANGKAT JARINGAN DI ITS

Nama : **AFIF RIDHO KAMAL PUTRA**
NRP : **05111440000173**
Jurusan : **Departemen Informatika FTIK**
Pembimbing I : **Royyana Muslim Ijtihadie S.Kom,
M.Kom., Ph.D**
Pembimbing II : **Bagus Jati Santoso, S.Kom., Ph.D**

Abstrak

Saat ini, dengan didukung oleh konsep SaaS (Software as a Service), aplikasi web berkembang dengan pesat. Para penyedia layanan aplikasi web berlomba-lomba memberikan pelayanan yang terbaik, seperti menjaga QoS (Quality of Service) sesuai dengan perjanjian yang tertuang dalam SLA (Service Level Agreement). Hal tersebut dikarenakan permintaan akses ke suatu aplikasi web biasanya meningkat dengan seiring berjalannya waktu. Keramaian akses sesaat menjadi hal yang umum dalam aplikasi web saat ini. Saat hal tersebut terjadi, aplikasi web akan di akses lebih banyak dari kebiasaan. Jika aplikasi web tersebut tidak menyediakan kemampuan untuk menangani hal tersebut, bisa menyebabkan aplikasi web tidak dapat berjalan dengan semestinya yang sangat merugikan pengguna.

Elastic cloud merupakan salah satu bagian dari komputasi awan yang sedang populer, dimana banyak riset dan penelitian yang berfokus di bidang ini. Elastic cloud bisa digunakan untuk menyelesaikan permasalahan di atas. Lalu sebuah perangkat lunak bernama Docker dapat diterapkan untuk mendukung elastic cloud.

Dalam tugas akhir ini akan dibuat sebuah rancangan sistem yang memungkinkan aplikasi web berjalan di atas Docker. Sistem ini bisa beradaptasi sesuai dengan kebutuhan dari aplikasi yang sedang berjalan. Jika aplikasi membutuhkan sumber daya tambahan, sistem akan menyediakan sumber daya berupa suatu container baru secara otomatis dan juga akan mengurangi penggunaan sumber daya jika aplikasi sedang tidak membutuhkannya. Dari hasil uji coba, sistem dapat menangani sampai dengan 57.750 request dengan error request yang terjadi sebesar 7.83%.

Kata-Kunci: aplikasi web, autoscale, docker, elastic cloud

IMPLEMENTATION OF PUBLISH/SUBSCRIBE ON DESIGN OF NETWORK MONITORING DEVICE SYSTEM IN ITS

Name : AFIF RIDHO KAMAL PUTRA
NRP : 05111440000173
Major : Informatics FTIK
**Supervisor I : Royyana Muslim Ijtihadie S.Kom,
M.Kom., Ph.D**
Supervisor II : Bagus Jati Santoso, S.Kom., Ph.D

Abstract

Nowdays, with the concept of SaaS (Software as a Service), web applications have developed a lot. Web service providers are competing to provide the best service, such as QoS (Quality of Service) requirements specified in the SLA (Service Level Agreement). The load of web applications usually very drastically along with time. Flash crowds are also very common in today's web applications world. When flash crowds happens, the web application will be accessed more than usual. If the web applications does not provide the ability to do so, it can make the web application not work properly which is very disadvantageous to the users.

Elastic cloud is one of the most popular part of cloud computing, with much researchs in this subject. Elastic clouds can be used to solve the above problems. Then a Docker can be applied to support the elastic cloud.

In this final task will be made an application system that allows web applications running on top of Docker. This system can adjust according to the needs of the running applications. If the application requires additional resources, the system will automatically supply the resources of a new container and will

also reduce resource usage if the application is not needing it. From the test results, the system can handle up to 57,750 requests and error ratio of 7.83%.

Keywords: *autoscale, docker, elastic cloud, web application*

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Alhamdulillahirabbil'alamina, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Implementasi *Publish/Subscribe* pada Rancang Bangun Sistem Monitoring Perangkat Jaringan di ITS**. 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 Departemen Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesaiannya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Bapak Royyana Muslim Ijtihadie S.Kom, M.Kom., Ph.D selaku pembimbing I yang telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
3. Bapak Bagus Jati Santoso, S.Kom., Ph.D selaku pembimbing II yang juga telah membantu, membimbing, dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS pada masa pengerjaan Tugas Akhir, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku koordinator TA, dan segenap dosen Teknik Informatika yang telah memberikan ilmu dan pengalamannya.

5. Serta semua pihak yang telah turut membantu penulis dalam bentuk apapun selama proses penyelesaian 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 2018

Afif Ridho Kamal Putra

DAFTAR ISI

ABSTRAK	vii
ABSTRACT	ix
Kata Pengantar	xi
DAFTAR ISI	xiii
DAFTAR TABEL	xvii
DAFTAR GAMBAR	xix
DAFTAR KODE SUMBER	xxi
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah	2
1.4 Tujuan	2
1.5 Manfaat	3
BAB II TINJAUAN PUSTAKA	5
2.1 <i>Publish/subscribe</i>	5
2.2 <i>Websocket</i>	5
2.3 SNMP	7
2.3.1 OID	9
2.4 Nagios	10
2.5 REST API	11
BAB III DESAIN DAN PERANCANGAN	13
3.1 Kasus Penggunaan	13
3.2 Arsitektur Sistem	15
3.2.1 Desain Umum Sistem	15
3.2.2 Desain REST API	16

3.2.3	Desain Publisher Server	17
3.2.4	Desain Pub/Sub Server	19
3.2.5	Desain Consumer pada Application Server dan Websocket	20
3.2.6	Desain Database Server	21
3.2.7	Desain Antarmuka	22
BAB IV	IMPLEMENTASI	27
4.1	Lingkungan Implementasi	27
4.2	Implementasi REST API	27
4.2.1	Pemasangan Python Flask dan Peewee . .	28
4.2.2	Implementasi Endpoint pada REST API .	28
4.3	Implementasi Publisher Server	32
4.3.1	Pemasangan Nagios Sebagai Pemantau dan Pengumpul Data Perangkat	33
4.3.2	Pengumpulan Data dan Pembuatan Script Pengiriman	33
4.4	Implementasi Pub/Sub Server	37
4.5	Implementasi Consumer pada Application Server dan Websocket	37
4.6	Implementasi Database Server	38
4.7	Implementasi Antarmuka	43
4.7.1	Daftar Aplikasi	43
4.7.2	Informasi Aplikasi	44
4.7.3	Daftar <i>Container</i>	45
4.7.4	Metrik Aplikasi	45
BAB V	PENGUJIAN DAN EVALUASI	47
5.1	Lingkungan Uji Coba	47
5.2	Skenario Uji Coba	48
5.2.1	Skenario Uji Coba Fungsionalitas	49
5.2.2	Skenario Uji Coba Performa	53
5.3	Hasil Uji Coba dan Evaluasi	55
5.3.1	Uji Fungsionalitas	55

5.3.2 Hasil Uji Performa	59
BAB VI PENUTUP	67
6.1 Kesimpulan	67
6.2 Saran	68
DAFTAR PUSTAKA	69
BAB A INSTALASI PERANGKAT LUNAK	71
BAB B KODE SUMBER	83
BIODATA PENULIS	85

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

3.1	Daftar Kode Kasus Penggunaan	14
3.1	Daftar Kode Kasus Penggunaan	15
4.1	Daftar Endpoint pada REST API	28
4.1	Daftar Endpoint pada REST API	29
4.1	Daftar Endpoint pada REST API	30
4.1	Daftar Endpoint pada REST API	31
4.1	Daftar Endpoint pada REST API	32
4.2	Rincian Tabel <i>users</i> pada Database	38
4.2	Rincian Tabel <i>users</i> pada Database	39
4.3	Rincian Tabel <i>devices</i> pada Database	39
4.3	Rincian Tabel <i>devices</i> pada Database	40
4.4	Rincian Tabel <i>OID</i> pada Database	41
4.5	Rincian Tabel <i>subscribe</i> pada Database	42
4.6	Rincian Tabel <i>subscribeoid</i> pada Database	42
5.1	Spesifikasi Komponen	47
5.2	IP dan Domain Server	48
5.3	Skenario Uji Mengelola Aplikasi Berbasis Docker	50
5.4	Skenario Uji Fungsionalitas Aplikasi Dasbor	52
5.5	Hasil Uji Coba Mengelola Aplikasi Berbasis Docker	56
5.6	Hasil Uji Fungsionalitas Aplikasi Dasbor	57
5.7	Jumlah <i>Request</i> ke Aplikasi	59
5.8	Jumlah <i>Container</i>	60
5.9	Kecepatan Menangani <i>Request</i>	61
5.10	Penggunaan CPU	62
5.11	Penggunaan <i>Memory</i>	63
5.12	<i>Error Ratio Request</i>	64

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

2.1	Model Komunikasi <i>Websocket</i>	7
2.2	Contoh <i>Object Identifier</i> (OID)	10
3.1	Diagram Kasus Penggunaan	13
3.2	Desain Umum Sistem	16
3.3	Desain REST API	17
3.4	Desain Publisher Server	18
3.5	Desain Pub/Sub Server	20
3.6	Ilustrasi Cara Kerja Queue dan Exchange	21
3.7	Desain Database	22
3.8	Desain Antarmuka Menampilkan Daftar Perangkat Yang Tersedia	23
3.9	Desain Antarmuka Menampilkan Rincian dari Perangkat Terkait	24
3.10	Desain Antarmuka Pemantauan Perangkat	24
4.1	Dasbor Daftar Aplikasi	43
4.2	Dasbor Informasi Aplikasi	44
4.3	Dasbor Daftar <i>Container</i>	45
4.4	Dasbor Matrik Aplikasi	46
5.1	Grafik Jumlah <i>Container</i>	60
5.2	Grafik Kecepatan Menangani <i>Request</i>	61
5.3	Grafik Penggunaan CPU	62
5.4	Grafik Penggunaan Memory	63
5.5	Grafik Error Ratio	64

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

IV.1 Perintah Mengumpulkan Data Perangkat dengan SNMP	33
IV.2 Pseudocode inisiasi Kelas Database	34
IV.3 Pseudocode Target <i>Thread</i> Untuk Mengambil Data Perangkat	35
IV.4 Pseudocode Pengiriman Data Dengan Pika	36
IV.5 Pseudocode Menjalankan Thread	36
A.1 Isi Berkas docker-compose.yml	73
A.2 Isi Berkas registry.conf	73
A.3 Isi Berkas confd.toml	77
A.4 Isi Berkas haproxy.cfg.tmpl	77
A.5 Isi Berkas haproxy.toml	79
B.1 Let's Encrypt X3 Cross Signed.pem	83

(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

Semakin berkembangnya teknologi informasi menuntut semakin banyaknya penggunaan perangkat berupa komputer atau perangkat jaringan. Setiap perangkat jaringan memiliki fungsi masing-masing. Sebagai contoh, salah satu fungsi yang dimiliki oleh router adalah untuk mendistribusikan alamat kepada tiap host agar tiap host dapat berkomunikasi satu sama lain, lalu ada pula switch yang memiliki fungsi utama yaitu menerima informasi dari berbagai sumber yang tersambung dengannya, kemudian menyalurkan informasi tersebut kepada pihak yang membutuhkannya saja.

Pada suatu organisasi yang besar, kebutuhan akan perangkat jaringan sangatlah besar, terutama dalam hal jumlah perangkat yang digunakan. Banyaknya jumlah perangkat jaringan yang digunakan otomatis membuat jumlah perangkat jaringan yang dipantau juga banyak jumlahnya. Dikarenakan banyaknya jumlah perangkat jaringan yang harus dipantau, seringkali para teknisi mengalami kesulitan dalam memantau kinerja dari tiap perangkat jaringan. Oleh karena itu, dibutuhkan sebuah sistem yang dapat memantau kinerja dari setiap perangkat jaringan yang terpasang.

Aplikasi yang dirancang pada tugas akhir ini, menghadirkan sebuah sistem yang dapat memantau seluruh perangkat jaringan yang terhubung dalam jaringan dengan metode publish/subscribe, sehingga setiap user nantinya dapat memilih perangkat jaringan apa saja yang ingin dipantau, dan dapat memilih informasi apa

saja yang ingin didapat kan dari tiap-tiap perangkat jaringan yang telah dipilih.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana cara membuat agen polling untuk mengambil data pada suatu perangkat jaringan?
2. Bagaimana cara mengimplementasikan publish/subscribe sebagai middleware?
3. Bagaimana cara menyaring informasi yang dipilih oleh subscriber?

1.3 Batasan Masalah

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

1. User hanya dapat memonitoring perangkat jaringan (server tidak termasuk).
2. Parameter untuk memonitor perangkat jaringan adalah ketersediaan dan beban yang ditampung.
3. Performa yang diukur adalah response time.

1.4 Tujuan

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

1. User hanya dapat memonitoring perangkat jaringan (server tidak termasuk).
2. Parameter untuk memonitor perangkat jaringan adalah ketersediaan dan beban yang ditampung.
3. Performa yang diukur adalah response time.

1.5 Manfaat

Manfaat dari pembuatan tugas akhir ini antara lain adalah:

1. Memonitor ketersediaan dan beban pada sebuah perangkat jaringan.
2. Memudahkan user untuk memonitoring perangkat jaringan yang diinginkan.

(Halaman ini sengaja dikosongkan)

BAB II

TINJAUAN PUSTAKA

2.1 *Publish/subscribe*

Publish/subscribe muncul sebagai paradigma komunikasi yang populer untuk sistem terdistribusi dalam skala yang besar. dalam *publish/subscribe* *consumer* akan berlangganan ke suatu *event* yang diinginkan. terlepas dari kegiatan *consumer*, ada *event producer* yang akan menerbitkan suatu *event*. jika *event* yang diterbitkan oleh produser cocok dengan *event* yang dilanggani oleh *consumer*, *event* tersebut akan dikirim kepada *consumer* secara *asynchronus*. Interaksi ini difasilitasi oleh *middleware publish/subscribe*. *middleware publish/subscribe* dapat dipusatkan menjadi sebuah *node* tunggal yang berperan sebagai *broker* dari sebuah *event* atau dipisahkan menjadi kumpulan beberapa *node broker* dari sebuah *event*.

pada dasarnya, *publish/subscribe* dibagi dari dua jenis yaitu: *topic-based* dan *content-based*. Pada *topic-based publish-subscribe*, *event* diterbitkan melalui sebuah topik dan *consumer event* akan berlangganan topik tersebut untuk mendapatkan data dari suatu *event*. berlangganan pada kasus *topic-based* tidak didukung pemilahan data dari suatu *event*. contohnya, *consumer* akan menerima semua data dari suatu *event* yang diterbitkan pada suatu topik. pada *content-based publish-subscribe*, berlangganan pada kasus ini didukung oleh fitur pemilahan yang diterapkan pada suatu *event* yang diterbitkan. data yang dipilah oleh *consumer* pada suatu *event* yang berlangganan akan dikirimkan ke *consumer*. [2]

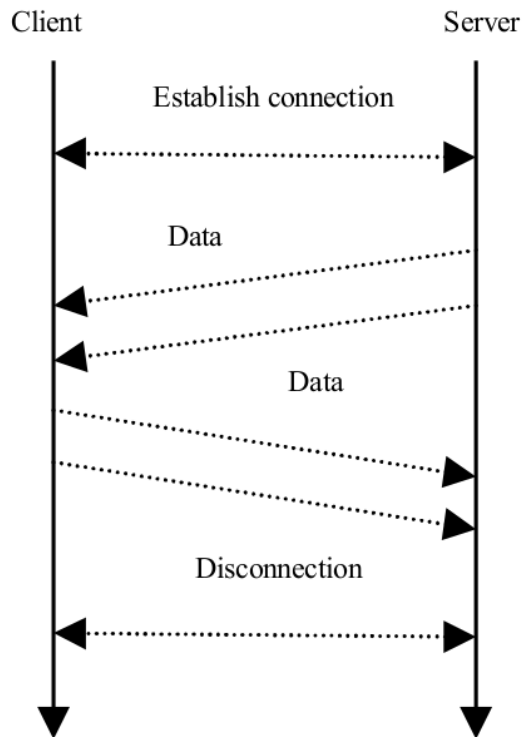
2.2 *Websocket*

Websocket adalah protokol berbasis TCP yang menyediakan channel komunikasi full-duplex antara server dan client melalui koneksi TCP tunggal. dibandingkan dengan skema komunikasi web real-time tradisional, protokol websocket menghemat

banyak sumber daya bandwidth pada jaringan, sumber daya server, dan performa real-time yang sangat jauh lebih baik dibanding websocket tradisional. Websocket adalah protocol berbasis TCP yang independen. Websocket hanya berhubungan dengan HTTP yang memiliki handshake yang diterjemahkan oleh HTTP server sebagai pengembangan dari sebuah request. Websocket terdiri dari dua bagian yaitu: handshake dan data transfer.

Untuk membuat koneksi Websocket, client harus mengirimkan request HTTP kepada server. setelah itu protokol akan diupgrade menjadi protokol Websocket. setelah itu server akan mengenali tipe request berdasarkan header pada HTTP. Protokol akan diupgrade menjadi Websocket apabila diminta oleh Websocket, dan kedua kubu (client dan server) akan memulai komunikasi full-duplex, yang berarti client dan server dapat bertukar data kapanpun sampai salah satu dari client atau server menutup koneksi tersebut. Model komunikasi Websocket dapat dilihat pada gambar 2.1

Websocket memiliki kemampuan yang lebih baik dalam berkomunikasi dibandingkan dengan skema komunikasi tradisional, dimana komunikasi terjadi secara realtime. sekali koneksi sudah berhasil dibuat, server dan client melakukan aliran data dua arah, dimana aktivitas tersebut meningkatkan kemampuan server untuk mengirim data. Bandingkan dengan protokol HTTP, dimana informasi yang dikirimkan lebih ringkas dan mengurangi transmisi dari data yang redundan. Dengan skala user yang besar dan kebutuhan komunikasi realtime yang tinggi, menurunkan beban pada jaringan akan menjadi keuntungan dibanding komunikasi realtime secara tradisional.



Gambar 2.1: Model Komunikasi *Websocket*

[3].

2.3 SNMP

Simple Network Management Protocol (SNMP) adalah aplikasi pada layer protokol yang digunakan untuk mengatur data pada jaringan. hampir semua vendor jaringan mendukung protokol SNMP. beberapa vendor peralatan telekomunikasi juga mulai didukung oleh protokol SNMP untuk mencapai pengaturan (manajemen) yang terintegrasi. Banyak dari aktivitas manajemen

jaringan pada jaringan enterprise yang menggunakan SNMP dalam persentasi yang sangat besar.

SNMP yang berdasarkan paradigma server - client termasuk kedalam manajemen stasiun, agen dan Management Information Bases (MIB). tujuan dari manajemen stasiun adalah untuk mengirimkan request kepada agent dan mengendalikan mereka, manajemen stasiun juga menyediakan antarmuka antara manajer jaringan manusia dan sistem manajemen jaringan. setiap perangkat jaringan memungkinkan untuk mempunyai agen yang dapat mengendalikan basis data dan ketika stasiun manajemen mulai melakukan polling, agen-agen tersebut akan mengirimkan laporan kepada stasiun manajemen.

Dalam pendekatan pemantauan dengan menggunakan SNMP, setiap agen akan mengirimkan stasiun manajemen sebuah informasi melalui polling laporan kejadian. Polling adalah aktivitas untuk melakukan interaksi antara agen dan stasiun manajemen menggunakan metode request dan response. Namun, stasiun manajemen hanya mendengarkan kepada informasi masuk pada pendekatan pelaporan kejadian. Agen akan mengirim informasi kepada stasiun manajemen setiap informasi tersebut dibutuhkan berdasarkan sebuah keputusan.

Pendekatan pemantauan secara realtime akan didefinisikan sebagai persetujuan antara agen dan stasiun manajemen dimana pada persetujuan semacam ini agen harus mengirimkan informasi kepada manajemennya secara berkala tanpa permintaan dari stasiun.

Dalam sebuah kelompok, status dan sifat dari sistem akan dipertimbangkan sebagai pekerjaan memantau dari informasi MIB. tipe data yang akan digunakan untuk memantau lebih penting dibandingkan dengan perancangan jaringan. berikut ini adalah informasi yang harus digunakan dalam pemantauan:

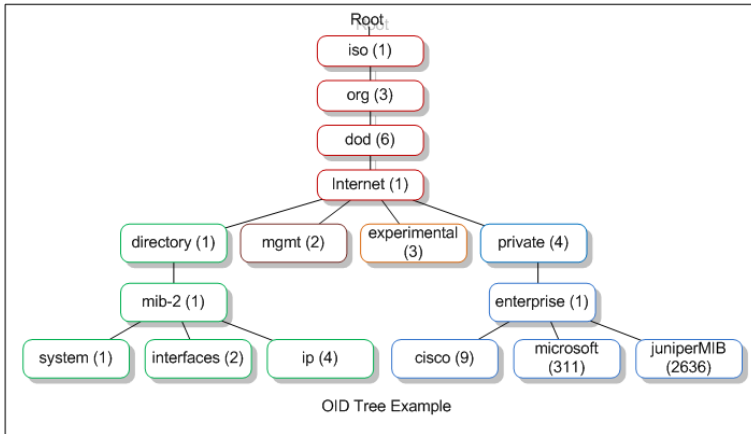
- Static: Struktur dan elemen didalam konfigurasi dikategorikan seperti id dari port pada sebuah router atau

host. Informasi ini akan jarang berubah.

- Dynamic: Informasi kejadian pada jaringan seperti paket dan elemen jaringan
- Statistical: Informasi harus berasal dari informasi dinamis seperti rata-rata dari paket yang dikirimkan pada tiap unit.

2.3.1 OID

Object Identifier adalah sesuatu untuk mengidentifikasi sebuah objek. Objek ini dapat berupa daerah atau disk drive tunggal. Yang paling umum, didalam IEEE-RAC, adalah OUI (Organizationally Unique Identifier), dan diturunkan secara terorganisir, dan terdaftar diluar OUI. pengidentifikasi yang paling umum selanjutnya, termasuk pengidentifikasi alamat ethernet adalah pengidentifikasi Extended Unique Identifiers (EUI) atau the World Wide Name (WWN). uniknya, untuk sistem yang sesuai, merupakan properti berharga dalam dua kasus ini. keunikan ini diasumsikan oleh struktur dari nomor unik yang dimulai dengan OUI. IEEE-RAC menetapkan OUI sebagai Object Identifier untuk sebuah organisasi. Object Identifier ini merupakan lapisan didalam konteks yang lebih luas dari pengidentifikasi yang diturunkan secara unik dari titik awal dari sebuah OID, International Telecommunication Union Telecommunication Standardization Sector (ITU-T) dan dideskripsikan didalam standar ASN.1. jalur tersebut dilacak menuju ITU-T disebut sebagai "arc" dari sebuah OID. arc ini berkembang menjadi OUI dan RAC lain menetapkan perancang dan melalui penempatan yang dibuat oleh organisasi hingga titik akhir dari sebuah Object Identifier.



Gambar 2.2: Contoh *Object Identifier* (OID)

2.4 Nagios

Nagios adalah perangkat lunak opensource yang aktif dikembangkan, memiliki banyak user juga komunitas yang luas, memiliki banyak plugin tambahan yang dikembangkan oleh user maupun yang terdapat langsung pada awal pengaturan, dan banyak buku tentang Nagios yang diterbitkan. Nagios juga merupakan sistem pemantauan yang paling populer yang cocok dengan hampir semua distribusi linux. Dukungan komersial tersedia dari perusahaan yang didirikan oleh penciptanya dan pengembang utama sebagai penyedia solusi resmi. Peralatan pemantauan yang berbasis nagios juga tersedia, seperti sensor yang dirancang untuk beroperasi bersama nagios. Karena fleksibilitas dari rancangan perangkat lunak yang menggunakan arsitektur plug-in, layanan pengecekan untuk aplikasi yang pustakanya sudah ditentukan dapat di gunakan. didalam nagios terdapat beberapa plugin lain, seperti script tambahan yang dapat dikostumisasi dan dapat digunakan pada nagios. Nagios adalah program yang ringan dan menyediakan

alat pemantauan yang sempurna yang dapat membantu untuk memantau seluruh protokol yang aktif dan perangkat jaringan yang terhubung dengan topologi. Nagios juga mampu untuk menyediakan grafik yang komperhensif dan bersifat realtime dan analisis tren.

2.5 REST API

(Halaman ini sengaja dikosongkan)

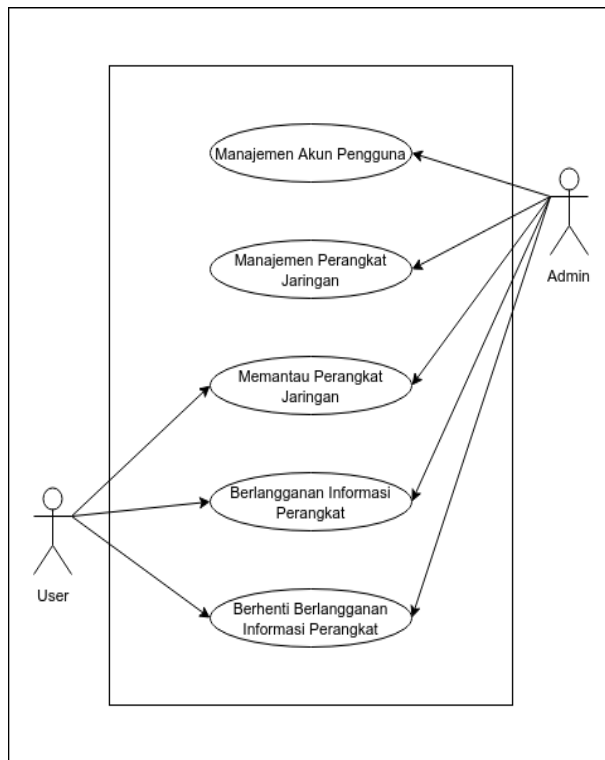
BAB III

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan sistem.

3.1 Kasus Penggunaan

Terdapat dua aktor dalam sistem ini, yaitu Pengembang (Administrator) dan *User* (Pengguna) dari aplikasi web yang dikelola oleh sistem. Diagram kasus penggunaan digambarkan pada Gambar 3.1.



Gambar 3.1: Diagram Kasus Penggunaan

Diagram 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	Manajemen Akun Pengguna.	Pengembang (Admin) dapat membuat, melihat, mengubah dan menghapus data akun pengguna.
UC-0002	Manajemen Perangkat Jaringan.	Pengembang (Admin) dapat membuat, melihat, mengubah dan menghapus data perangkat jaringan.
UC-0003	Memantau Perangkat Jaringan.	Pengembang (Admin) dan Pengguna (User) dapat memantau seluruh perangkat jaringan yang sudah ia langgani.
UC-0004	Berlangganan Informasi Perangkat.	Pengembang (Admin) dan Pengguna (User) dapat berlangganan informasi perangkat jaringan yang diinginkan.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-0005	Berhenti Berlangganan Informasi Perangkat.	Pengembang (Admin) dan Pengguna (User) dapat berhenti berlangganan informasi perangkat jaringan yang diinginkan.

3.2 Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis dan kebutuhan bisnis dan desain dari sistem yang akan dibangun.

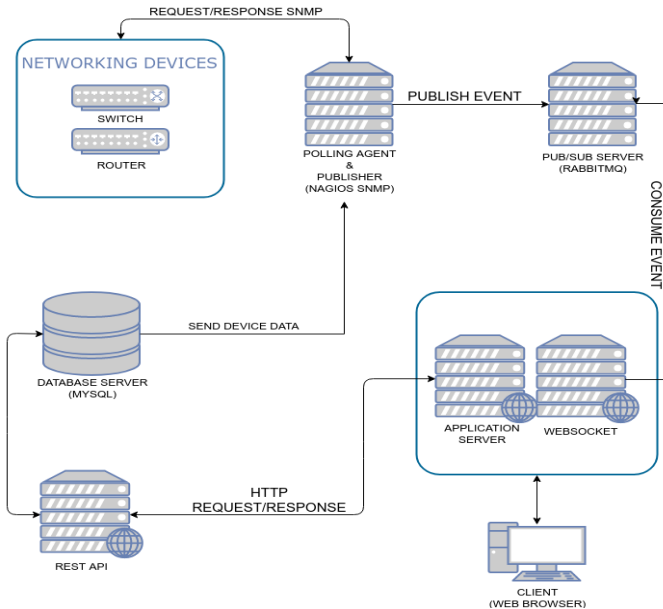
3.2.1 Desain Umum Sistem

Sistem yang akan dibuat yaitu sistem yang dapat melakukan pemantauan pada perangkat jaringan yang berbasis *web* dengan metode *publish/subscribe*, dimana pengguna (*user*) harus berlangganan kepada suatu informasi untuk mendapatkan informasi yang diinginkan.

Sistem ini melibatkan 3 (Tiga) server yang berfungsi sebagai web server dan 1 (satu) server yang berfungsi sebagai database server. Server aplikasi dan websocket server berada pada satu server, sehingga pada implementasinya webserver aplikasi dan websocket dijalankan pada port yang berbeda. Pada sistem ini *client* yaitu pengguna (*user*) dan pengelola (*admin*) akan mengakses aplikasi menggunakan web browser. yang nantinya jika mengakses fitur selain memantau perangkat jaringan, aplikasi akan mengirimkan *request* HTTP kepada REST API, dimana REST API tersebut melakukan transaksi data kepada

database server. setelah itu REST API akan mengirimkan *response* kepada aplikasi.

jika client mengakses fitur memantau jaringan, maka aplikasi akan terhubung dengan websocket yang tugasnya mengakses data yang berada pada pub/sub server, dimana pubsub server menyimpan data yang diterbitkan oleh nagios, data tersebut adalah hasil response SNMP nagios kepada tiap perangkat jaringan terkait. Penjelasan secara umum arsitektur sistem akan diuraikan pada Gambar 3.2.



Gambar 3.2: Desain Umum Sistem

3.2.2 Desain REST API

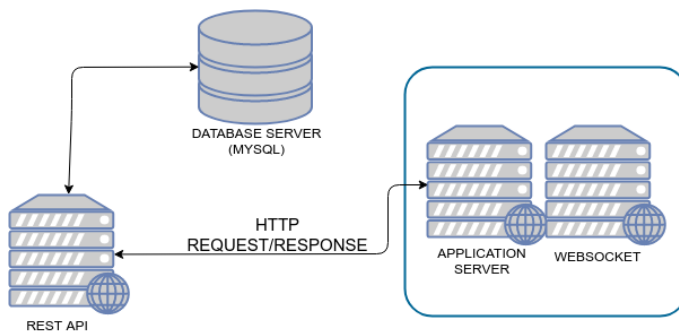
REST API bertujuan untuk menjadikan sistem yang memiliki performa yang baik, cepat dan mudah untuk di kembangkan

(scaleable) terutama dalam pertukaran dan komunikasi data. REST API diakses menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten akan menghasilkan API yang baik dan mudah untuk dimengerti developer. URL API biasa disebut endpoint dalam pemanggilannya.

Pada sistem ini terdapat beberapa endpoint, beberapa endpoint dibagi menjadi beberapa endpoint sesuai dengan perintah yang dijalankannya. misal: create, read, delete, update dan lain-lain.

Server aplikasi mengirimkan HTTP request kepada REST API yang nantinya REST API akan melakukan transaksi data pada database sesuai dengan endpointnya masing-masing. setelah itu REST API akan mengirimkan HTTP response kepada server aplikasi.

Secara umum, arsitektur dari REST API dapat dilihat pada Gambar 3.3



Gambar 3.3: Desain REST API

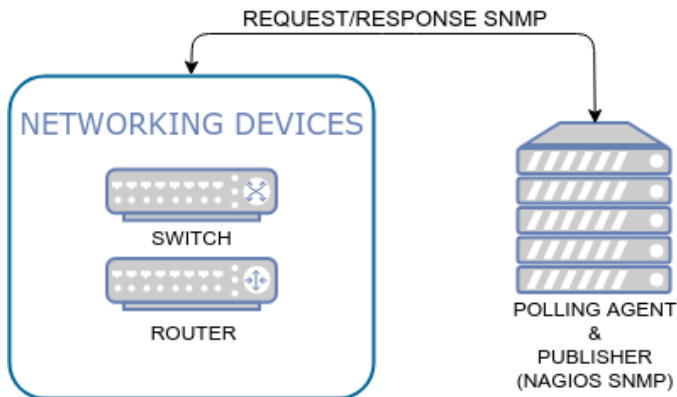
3.2.3 Desain Publisher Server

Pada publisher server, dipasang aplikasi untuk memantau kinerja jaringan, yaitu Nagios. Pada nagios, terdapat plugin

untuk memantau kinerja jaringan dengan protokol SNMP yaitu `check-snmp`. plugin ini membutuhkan beberapa parameter, diantaranya: alamat perangkat yang ingin dipantau dan oid dari apa yang ingin dipantau.

Sebuah script dibuat untuk mengambil data dan mengirimkannya menuju pub/sub server. setiap perangkat yang dipantau dimasukkan ke sebuah thread baru agar dapat berjalan secara paralel. didalam thread tersebut, setiap perangkat terkait diperiksa kinerjanya dengan protokol SNMP dan hasilnya dikirimkan kepada pub/sub server melalui sebuah exchange yang telah diikat dengan sebuah message queue yang sebelumnya telah diinisiasi. Rancangan umum dari *Publisher Server* seperti yang digambarkan pada Gambar 3.4.

Exchange yang dibuat oleh script tersebut namanya dibuat berdasarkan uuid versi 4 dari tiap device yang diambil dari database dan nama queue dibuat berdasarkan uuid versi 4 yang dibuat baru.



Gambar 3.4: Desain Publisher Server

3.2.4 Desain Pub/Sub Server

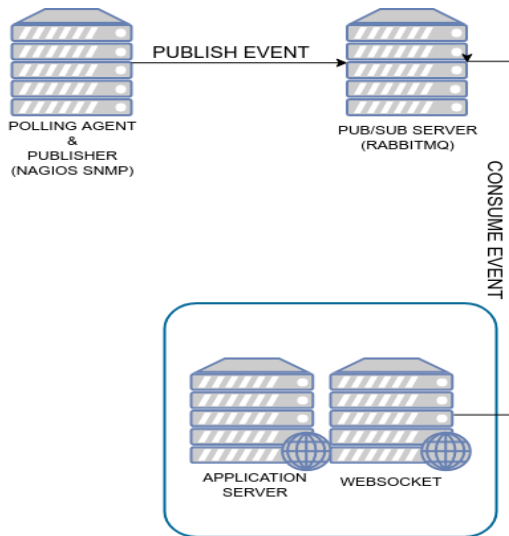
Publish/subscribe server atau bisa juga disebut pub/sub server. yaitu sebuah server untuk menampung seluruh pesan yang dikirimkan oleh publisher. didalamnya terpasang aplikasi message broker yaitu `RabbitMQ`. seluruh pesan yang dikirimkan oleh publisher dikirimkan ke pub/sub server melalui sebuah exchange yang diikat dengan sebuah queue setelah itu server akan menyimpan pesan queue tersebut hingga ada consumer yang meminta data tersebut untuk dikirimkan. dalam kasus ini yang bertindak sebagai consumer adalah websocket server.

Di sisi websocket dan server aplikasi, websocket menginisiasi sebuah exchange yang namanya dibuat berdasarkan uuid versi 4 dari tiap perangkat yang ingin dipantau dari database server, dengan syarat exchange dengan nama tersebut belum dibuat atau terdaftar sebelumnya. jika exchange dengan nama tersebut sudah dibuat atau terdaftar sebelumnya pada pub/sub server maka websocket server tidak perlu membuat exchange tersebut.

Begitu juga dengan queue-nya. queue dibuat dengan nama uuid yang telah dibuat acak oleh client dengan algoritma uuid versi 4, dengan syarat queue dengan nama tersebut belum dibuat atau terdaftar sebelumnya. Jika queue dengan nama tersebut sudah dibuat atau terdaftar sebelumnya pada pub/sub server maka websocket server tidak perlu membuat queue tersebut.

Setelah menghadapi masalah pembuatan exchange dan queue, websocket baru mengambil data perangkat pada pub/sub server sesuai dengan data apa saja yang dilangani oleh client.

Secara umum, arsitektur rancangan dari *Pub/Sub Server* dapat dilihat pada Gambar 3.5.



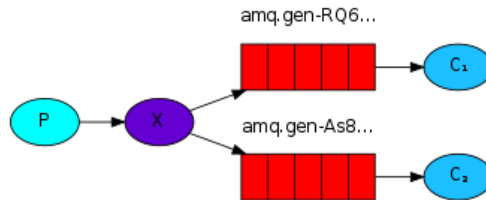
Gambar 3.5: Desain Pub/Sub Server

3.2.5 Desain Consumer pada Application Server dan WebSocket

Consumer berfungsi untuk mengambil data yang dibutuhkan oleh client dari pub/sub server. Pada sistem ini, consumer didesain untuk diimplementasikan pada websocket agar data yang diterima oleh client adalah data yang paling terbaru (realtime). websocket ini nantinya akan disambungkan dengan sebuah endpoint (URL) pada aplikasi.

Consumer ini nantinya akan membuat sebuah queue dengan nama yang ditentukan oleh client. nama dari queue tersebut ditentukan dengan membuat string UUID versi 4 secara acak. Setelah berhasil membuat queue, consumer membuat exchange yang banyaknya sejumlah perangkat yang terdaftar pada sistem dan exchange tersebut diberi nama sesuai dengan ID pada masing-masing perangkat yang dimana ID tersebut berformat

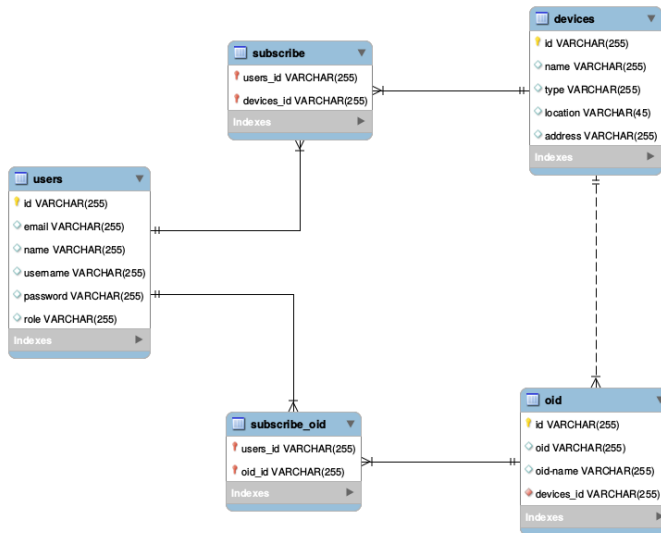
UUID versi 4. Pembuatan queue dan exchange akan dilakukan jika queue dan exchange belum terdaftar pada pub/sub server. jika queue dan exchange sudah terdaftar, maka tidak akan ada queue dan exchange yang akan dibuat. Setelah queue dan exchange berhasil dibuat. queue tersebut akan diikat dengan satu atau lebih exchange. lewat queue tersebutlah data akan dikirim. ilustrasi cara kerja queue dan exchange dapat dilihat pada gambar 3.6.



Gambar 3.6: Ilustrasi Cara Kerja Queue dan Exchange

3.2.6 Desain Database Server

Desain database pada sistem ini adalah seperti yang digambarkan pada gambar 3.7. Terdapat tiga tabel utama yang mewakili tiap entitas yang terlibat dalam sistem ini, yaitu: users, devices, dan oid. selain itu, terdapat dua table many-to-many untuk menyimpan data pengguna yang telah berlangganan kepada tiap perangkat dan pengguna yang berlangganan kepada tiap OID (untuk mengetahui informasi apa saja yang ada pada tiap perangkat. tiap OID memiliki informasi yang berbeda).



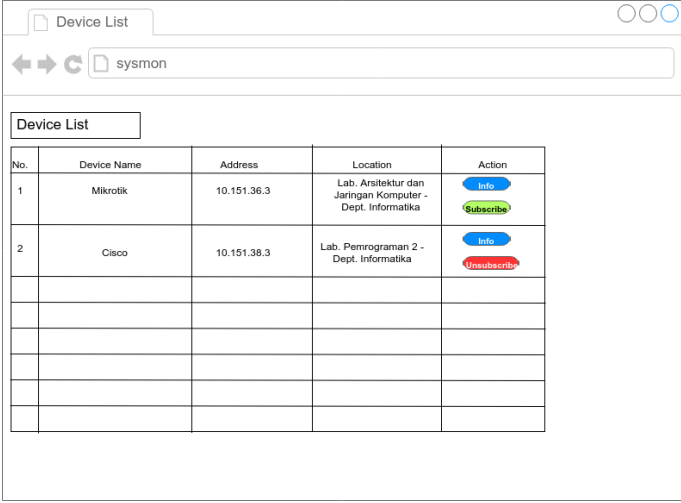
Gambar 3.7: Desain Database

3.2.7 Desain Antarmuka

Desain antarmuka adalah desain untuk halaman yang nantinya akan digunakan oleh client baik itu pengguna (user) ataupun pengelola (admin). Antarmuka yang nantinya dibuat berbasis web dan menggunakan Bootstrap 3 dan HTML. terdapat beberapa perbedaan pada antarmuka yang digunakan oleh pengelola dan pengguna. Misal, pada antarmuka yang digunakan pengguna tidak ada tombol untuk menghapus data perangkat, sedangkan pada antarmuka yang digunakan oleh pengelola terdapat tombol untuk menghapus data perangkat yang telah terdaftar.

Desain antarmuka untuk menampilkan daftar seluruh perangkat yang tersedia pada sistem dapat dilihat pada gambar 3.8. pada halaman ini pengguna dan pengelola dapat melihat daftar perangkat yang tersedia dan beberapa infonya, seperti:

nama perangkat, alamat, dan lokasi dari perangkat tersebut. pada halaman ini pengguna dan pengelola juga bisa langsung berlangganan atau berhenti berlangganan dengan menekan sebuah tombol yang ada pada halaman ini.

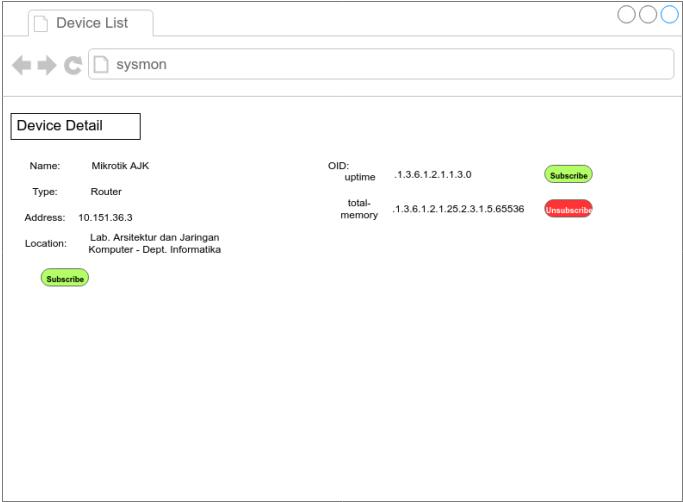


No.	Device Name	Address	Location	Action
1	Mikrotik	10.151.38.3	Lab. Arsitektur dan Jaringan Komputer - Dept. Informatika	Info Subscribe
2	Cisco	10.151.38.3	Lab. Pemrograman 2 - Dept. Informatika	Info Unsubscribe

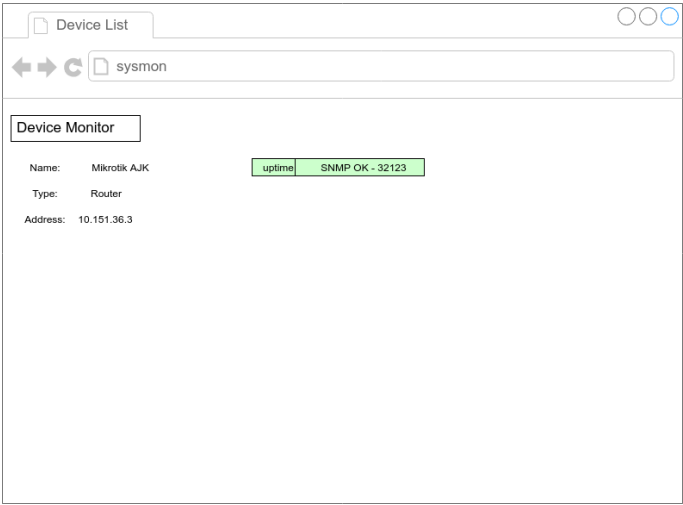
Gambar 3.8: Desain Antarmuka Menampilkan Daftar Perangkat Yang Tersedia

Desain antarmuka untuk menampilkan rincian dari antarmuka terkait dapat dilihat pada gambar 3.9. pada halaman ini pengguna dan pengelola dapat melihat seluruh rincian data yang ada pada perangkat. mulai dari nama perangkat, tipe perangkat, alamat perangkat, lokasi perangkat dan info apa saja yang dapat dipantau melalui OID.

Pada halaman ini pengguna dan pengelola juga dapat berlangganan dengan cara menekan sebuah tombol. tidak hanya berlangganan perangkatnya saja, pengguna dan pengelola juga dapat memilih untuk berlangganan info apa saja yang ingin didapatkan dari perangkat tersebut.



Gambar 3.9: Desain Antarmuka Menampilkan Rincian dari Perangkat Terkait



Gambar 3.10: Desain Antarmuka Pemantauan Perangkat

Desain antarmuka pemantauan perangkat dapat dilihat pada gambar 3.10. pada halaman ini, pengguna dan pengelola akan mendapatkan data dari seluruh perangkat yang sudah dilanggan. data yang ditampilkan pada halaman ini dipilih berdasarkan info yang dipilih pada antarmuka menampilkan rincian dari antarmuka terkait yang bisa dilihat pada gambar 3.9.

(Halaman ini sengaja dikosongkan)

BAB IV

IMPLEMENTASI

Bab ini membahas implementasi sistem Pengendali Elastisitas secara rinci. Pembahasan dilakukan secara rinci untuk setiap komponen yang ada, yaitu: *docker registry*, *master host*, *controller*, *load balancer*, dan dasbor.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan virtualisasi Proxmox dengan spesifikasi Host komputer adalah Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz dengan memori 8 GB di Laboratorium Arsitektur dan Jaringan Komputer, Teknik Informatika ITS. Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut:

- Sistem Operasi Linux Ubuntu Server 16.04 LTS
- RabbitMQ 3.7.5-1
- MySQL Ver 15.1 Distrib 10.0.34-MariaDB
- Python 2.7
- Flask 0.12.2
- Node.js v6.11.4
- Nagios 4.3.4
- Express.js 4.16.3

4.2 Implementasi REST API

REST API digunakan untuk memudahkan aplikasi agar ringan dan mudah untuk dikembangkan. pada tugas akhir ini, REST API memiliki fungsi utama untuk menyimpan data user yang berlangganan pada perangkat jaringan atau berlangganan OID (Informasi didalam perangkat jaringan). REST API dibangun dengan framework Python yaitu Flask dan dilengkapi ORM (Object-relational mapping) Database yaitu Peewee.

4.2.1 Pemasangan Python Flask dan Peewee

Pemasangan `Python Flask` dapat dilakukan dengan mudah, cukup dengan memasangnya dengan manajer paket yang dimiliki oleh `Python` yaitu `Pip`. Setelah `Flask` berhasil terpasang, selanjutnya adalah tahap pemasangan ORM `Peewee`. `Peewee` adalah Object-relational Mapping dimana fungsi utamanya adalah memudahkan pengembang agar dapat menyambungkan aplikasi dengan database dan melakukan query dengan mudah. pemasangan ORM `Peewee` dapat dilakukan dengan cara mengambil berkas instalasinya pada git <https://github.com/coleifer/peewee.git> dan pasang `Peewee` sesuai dengan instruksi yang tertera pada situs git tersebut.

4.2.2 Implementasi Endpoint pada REST API

REST API diakses menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten akan menghasilkan API yang baik dan mudah untuk dimengerti developer. URL API biasa disebut endpoint dalam pemanggilannya.

Pada sistem ini terdapat beberapa endpoint, beberapa endpoint dibagi menjadi beberapa endpoint sesuai dengan perintah yang diajalankannya. misal: create, read, delete, update dan lain-lain.

Berikut ini adalah endpoint yang dibuat dalam sistem ini:

Tabel 4.1: Daftar Endpoint pada REST API

No	Endpoint (Route)	Metode	Aksi
1	/register	POST	Membuat data baru pada tabel user di database

Tabel 4.1: Daftar Endpoint pada REST API

No	Endpoint (Route)	Metode	Aksi
2	/login	POST	Mengambil data pada tabel user dan mencocokkannya dengan JSON yang dikirimkan lewat body. setelah data username dan password cocok, lalu dibuatkan sebuah token JWT.
3	/logout	POST	Memasukkan token JWT yang terdaftar pada server kedalam daftar hitam agar token tidak dapat digunakan lagi.
4	/users	GET	Menampilkan seluruh data user yang terdaftar pada sistem

Tabel 4.1: Daftar Endpoint pada REST API

No	Endpoint (Route)	Metode	Aksi
5	/users/<string:username>	GET	Menampilkan data user berdasarkan username yang tertulis pada URL
6	/devices/create	POST	Membuat data baru pada tabel devices di database
7	/devices/edit/<string:id>	PUT	Mengubah data pada tabel devices di database yang ID nya sama dengan ID yang ada pada URL.
8	/devices/delete	DELETE	Menghapus data pada tabel devices di database yang ID nya tertulis pada body yang bertipe JSON.
9	/devices	GET	Menampilkan seluruh data perangkat yang terdaftar pada sistem

Tabel 4.1: Daftar Endpoint pada REST API

No	Endpoint (Route)	Metode	Aksi
10	/devices/<string:id>	GET	Menampilkan data user berdasarkan username yang tertulis pada URL
11	/oid/create	POST	Membuat data baru pada tabel oid
12	/oid/edit	POST	Mengubah data pada tabel oid di database yang ID nya tertulis pada body yang bertipe JSON.
13	/oid/delete	POST	Menghapus data pada tabel oid di database yang ID nya tertulis pada body yang bertipe JSON.
14	/subscribe/devices	POST	Membuat data baru pada tabel subscribe

Tabel 4.1: Daftar Endpoint pada REST API

No	Endpoint (Route)	Metode	Aksi
15	/unsubscribe/devices	POST	Menghapus data pada tabel subscribe di database yang ID nya tertulis pada body yang bertipe JSON.
16	/subscribe/oid	POST	Membuat data baru pada tabel subscribe_oid
17	/unsubscribe/oid	POST	Menghapus data pada tabel subscribe_oid di database yang ID nya tertulis pada body yang bertipe JSON.

4.3 Implementasi Publisher Server

Publisher server merupakan server yang berfungsi untuk mengambil data pada perangkat jaringan secara berkala dan mengirimkannya menuju pub/sub server. publisher server menggunakan plugin `check_snmp` bawaan program Nagios, Sehingga untuk melakukan pengambilan data, kita perlu memasang nagios pada server.

setelah data berhasil dikumpulkan, data yang diambil pada tiap perangkat dikirimkan menuju pub/pub server melalui thread yang berbeda. proses ini dinamakan `multithreading`.

4.3.1 Pemasangan Nagios Sebagai Pemantau dan Pengumpul Data Perangkat

Pemasangan Nagios dapat dilakukan dengan beberapa cara, namun cara yang dipakai pada kasus ini adalah memasang Nagios langsung dari sumbernya untuk mendapatkan fitur terbaru, pembaharuan keamanan, dan pembetulan bug.

Berikut ini adalah sumber untuk mendapatkan nagios yang siap untuk dipasang: <https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.3.4.tar.gz>

Nagios perlu beberapa perintah khusus yang hanya bisa dilakukan oleh user yang bernama "nagios" maka dari itu diperlukan user pada server yang bernama "nagios" dengan nama group "nagcmd". Selain user, Nagios juga perlu beberapa paket yang harus terpasang sebelum memasang nagios itu sendiri. Beberapa paket diantaranya adalah: build-essential, libgd2-xpm-dev, openssl, libssl-dev, unzip

Setelah Nagios terpasang, direktori kerja dari Nagios dapat dilihat pada direktori `/usr/local/nagios`

4.3.2 Pengumpulan Data dan Pembuatan Script Pengiriman

Untuk mengumpulkan data perangkat jaringan, dibutuhkan plugin bawaan nagios yang bernama `check_snmp`. plugin tersebut berada pada direktori `/usr/local/nagios/libexec` untuk menjalankan plugin tersebut dibutuhkan dua parameter, yaitu: alamat dari perangkat jaringan yang ingin dipantau dan OID dari data yang ingin didapatkan dari perangkat. perintah yang dijalankan untuk mendapatkan data pada perangkat jaringan lewat protokol SNMP adalah seperti yang tertulis pada kode sumber

```
$ /usr/local/nagios/check_snmp -H <
    alamat_perangkat> -o <oid_perangkat>
```

Kode Sumber IV.1: Perintah Mengumpulkan Data Perangkat dengan SNMP

Setelah data dapat dikumpulkan, sebuah script diperlukan untuk mengirim data tersebut menuju pub/sub server yang didukung oleh RabbitMQ sebagai Message Broker.

Sebuah library bernama `pika` dibutuhkan untuk mengirim data tersebut ke pub/sub server. tiap perangkat yang dikumpulkan datanya dan dikirimkan ke pub/sub server, diproses didalam sebuah thread yang berbeda. oleh karena itu inisiasi database dibutuhkan pada awal script untuk mengetahui ada berapa perangkat yang terdaftar pada sistem.

pertama-tama, masukkan library yang dibutuhkan untuk pembuatan script (termasuk `pika`), lalu dilanjutkan dengan potongan kode untuk menginisiasi database. Pseudocode untuk inisiasi kelas database dapat dilihat pada kode sumber IV.2

```
class BaseModel(Model):
class Meta:
    database = database

class Users(BaseModel):
    id = UUIDField(primary_key=True)
    name = CharField()
    username = CharField(unique=True)
    password = CharField()
    email = CharField()
    role = CharField()

class Devices(BaseModel):
    id = UUIDField(primary_key=True)
    name = CharField()
    type = CharField()
    location = CharField()
    address = CharField()

class Oid(BaseModel):
```

```

id = UUIDField(primary_key=True)
oid = CharField()
oidname = CharField()
devices_id = ForeignKeyField(Devices,
    on_delete='CASCADE')

class Subscribe(BaseModel):
    users_id = ForeignKeyField(Users, on_delete
        ='CASCADE')
    devices_id = ForeignKeyField(Devices,
        on_delete='CASCADE')

```

Kode Sumber IV.2: Pseudocode inisiasi Kelas Database

Setelah itu buat fungsi sebagai target menjalankan thread, nantinya tiap thread akan mengeksekusi kode yang ada didalam fungsi tersebut. didalam fungsi tersebut meliputi pegumpulan data dengan `check_snmp`. Data perangkat jaringan yang dikumpulkan dengan `check_snmp` dimasukkan kedalam sebuah python dictionary yang nantinya dictionary tersebut akan dikirimkan menuju pub/sub server. Pseudocode fungsi tersebut dapat dilihat pada kode sumber IV.3

```

rabbitMq(exchange, address):
    try:
        add getOidData() into array
            of dictionary
    except:
        add NULL into array of
            dictionary

    try:
        add getSnmpDeviceData()
            into JSON
    except:

```

add Error Message into JSON

Kode Sumber IV.3: Pseudocode Target *Thread* Untuk Mengambil Data Perangkat

Untuk mengirimkan data menuju pub/sub server diperlukan library pika yang akan membuat koneksi dengan RabbitMQ yang berada di pub/sub server. potongan kode untuk mengirimkan data tersebut dapat dilihat pada kode sumber IV.4

```
pika.openConnection()
if exchange does not exist:
    createExchage()
    if queue does not exist:
        createQueue()
        bindExchangetoQueue()
    else:
        bindExchangetoQueue()
else:
    pass
sendMessage()
```

Kode Sumber IV.4: Pseudocode Pengiriman Data Dengan Pika

Setelah seluruh fungsi selesai dibuat, langkah terakhir adalah membuat thread agar tiap thread nantinya akan menjalankan fungsi yang telah dibuat dan menjalankannya secara berkala. Pseudocode untuk membuat thread dapat dilihat pada kode sumber IV.5

```
Thread
while true:
    getDeviceId() as exchangename
    getDeviceAddress as deviceaddress
    thread(target=rabbitmq(), argument
           =(exchangename, deviceaddress))
    sleep(2)
```

Kode Sumber IV.5: Pseudocode Menjalankan Thread

4.4 Implementasi Pub/Sub Server

Pada pub/sub server, dipasang aplikasi message broker RabbitMQ. pada kasus ini RabbitMQ menerima seluruh data yang dikirimkan oleh publisher. Setelah itu RabbitMQ menyimpannya dan menunggu hingga ada consumer yang meminta data pada RabbitMQ. kriteria data yang dikirimkan harus dispesifikkan sesuai dengan apa yang diminta oleh consumer.

Pemasangan aplikasi RabbitMQ membutuhkan bahasa pemrograman erlang. untuk itu sebelum memasang RabbitMQ, harus terlebih dahulu memasang erlang pada sistem. Selain erlang, beberapa paket juga harus terpasang pada sistem, beberapa diantaranya adalah: `init-system-helpers`, `socat`, `adduser`, `logrotate`

Setelah RabbitMQ server terpasang, selanjutnya dibutuhkan sebuah web admin untuk RabbitMQ agar mudah untuk melakukan manajemen data, user dan lain-lain pada web admin tersebut. RabbitMQ sudah menyediakan *plugin* agar web admin dapat langsung digunakan. hanya dengan menjalankan perintah untuk mengaktifkan web admin dengan `rabbitmqctl`

4.5 Implementasi Consumer pada Application Server dan Websocket

Sebagai media penyimpan

4.6 Implementasi Database Server

Sebagai media penyimpanan, sebuah database diperlukan untuk menyimpan data pengguna, perangkat, dan data berlangganan. Terdapat tiga tabel utama yang mewakili tiap entitas yang terlibat dalam sistem ini, yaitu: `users`, `devices`, dan `oid`. selain itu, terdapat dua table `many-to-many` untuk menyimpan data pengguna yang telah berlangganan kepada tiap perangkat dan pengguna yang berlangganan kepada tiap OID (untuk mengetahui informasi apa saja yang ada pada tiap perangkat. tiap OID memiliki informasi yang berbeda).

Pada Tugas Akhir ini, sistem basis data yang digunakan adalah `Mysql Server` yang dimana `Mysql Server` termasuk kedalam RDBMS (*Relational Database Management System*). Berikut adalah rincian dari tabel yang diimplementasikan. rincian tabel `users` dapat dilihat pada tabel 4.2

Tabel 4.2: Rincian Tabel `users` pada Database

No	Kolom	Type Data	Keterangan
1	id	varchar(255)	Sebagai primary key pada tabel, nilai pada kolom ini berformat UUID versi 4
2	name	varchar(255)	Data yang berbentuk string. Digunakan untuk kelengkapan profil pengguna.
3	username	varchar(255)	Data yang berbentuk string. Digunakan untuk keperluan autentikasi.

Tabel 4.2: Rincian Tabel `users` pada Database

No	Kolom	Tipe Data	Keterangan
4	password	varchar(255)	Data yang berbentuk string, implementasinya berupa hash. Digunakan untuk keperluan autentikasi.
5	email	varchar(255)	Data yang berbentuk string. Digunakan untuk kelengkapan profil pengguna
6	role	varchar(255)	Data yang berbentuk string. Digunakan untuk kelengkapan profil pengguna dan pembeda peran agar setiap user memiliki hak istimewa masing-masing.

Tabel 4.3: Rincian Tabel `devices` pada Database

No	Kolom	Tipe Data	Keterangan
1	id	varchar(255)	Sebagai primary key pada tabel, nilai pada kolom ini berformat UUID versi 4

Tabel 4.3: Rincian Tabel `devices` pada Database

No	Kolom	Tipe Data	Keterangan
2	name	varchar(255)	Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan.
3	type	varchar(255)	Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan.
4	location	varchar(255)	Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan.
5	address	varchar(255)	Data yang berbentuk string, implementasinya berbentuk alamat IP dari tiap perangkat jaringan. Digunakan untuk mengkoleksi data pada publisher server.

Tabel 4.4: Rincian Tabel `OID` pada Database

No	Kolom	Tipe Data	Keterangan
1	id	varchar(255)	Sebagai primary key pada tabel, nilai pada kolom ini berformat UUID versi 4
2	oid	varchar(255)	Data yang berbentuk string, implementasinya berbentuk OID (Object-Identfier). Digunakan mengkoleksi perangkat jaringan pada publisher server.
3	oidname	varchar(255)	Data yang berbentuk string. Digunakan untuk kelengkapan profil dari perangkat jaringan.
4	devices_id	varchar(255)	Merupakan foreign key dari id pada tabel devices. Data ini berbentuk string, nilai pada kolom ini berformat UUID versi 4.

Tabel 4.5: Rincian Tabel `subscribe` pada Database

No	Kolom	Tipe Data	Keterangan
1	users_id	varchar(255)	Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel users, nilai pada kolom ini berformat UUID versi 4
2	devices_id	varchar(255)	Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel devices, nilai pada kolom ini berformat UUID versi 4.

Tabel 4.6: Rincian Tabel `subscribeoid` pada Database

No	Kolom	Tipe Data	Keterangan
1	users_id	varchar(255)	Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel users, nilai pada kolom ini berformat UUID versi 4

Tabel 4.6: Rincian Tabel `subscribeoid` pada Database

No	Kolom	Tipe Data	Keterangan
2	oid_id	varchar(255)	Sebagai primary key pada tabel juga sebagai foreign key dari id pada tabel <code>oid</code> , nilai pada kolom ini berformat UUID versi 4.

4.7 Implementasi Antarmuka

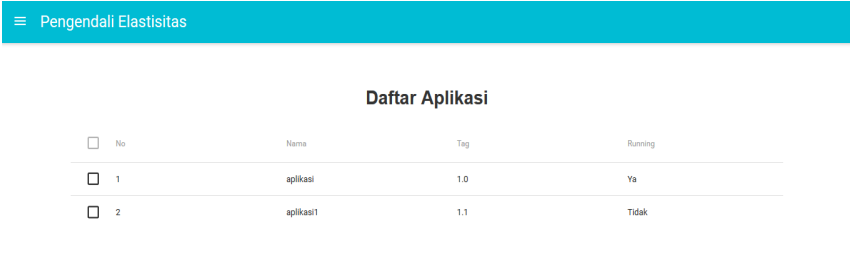
Dasbor diimplementasikan menggunakan perangkat kerja React bagian *frontend* dan Flask untuk *backendnya*. Dasbor digunakan untuk mempermudah pengembang dalam mengelola aplikasi. Dasbor memiliki menu-menu sebagai berikut:

- Daftar Aplikasi
- Informasi Aplikasi
- Daftar *Container*
- Matrik Aplikasi

Masing-masing menu berikutnya akan dijelaskan secara rinci.

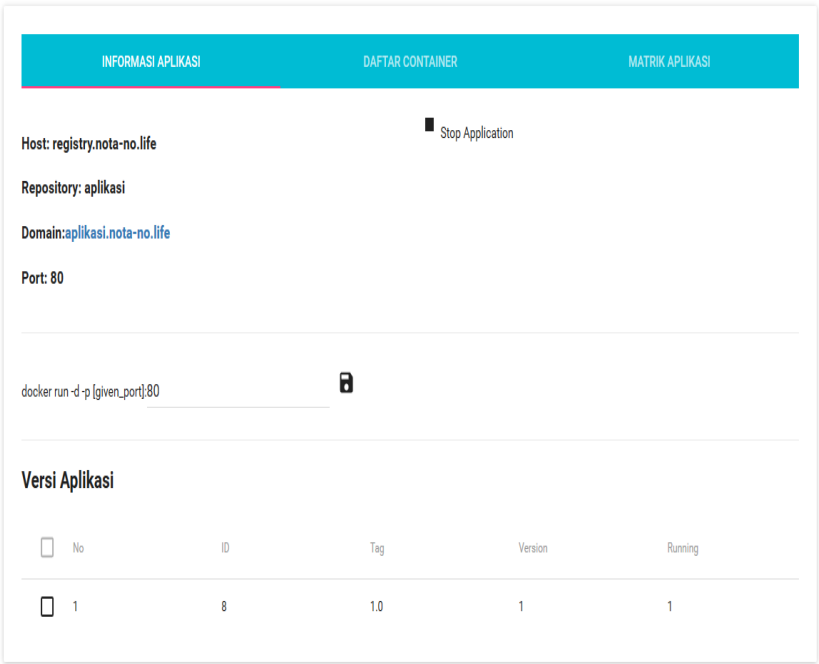
4.7.1 Daftar Aplikasi

Daftar aplikasi, juga merupakan beranda dari dasbor, adalah menu yang digunakan untuk melihat daftar aplikasi atau *image* yang ada pada *server docker registry*. Pada halaman ini, bisa dilihat nama beserta versi terakhir dari aplikasi. Lalu juga terdapat status apakah aplikasi sedang berjalan atau tidak. Antar muka kelola daftar aplikasi ditunjukkan pada Gambar 4.1.



Gambar 4.1: Dasbor Daftar Aplikasi

4.7.2 Informasi Aplikasi



Gambar 4.2: Dasbor Informasi Aplikasi

Halaman ini menunjukkan informasi lengkap dari sebuah aplikasi. Pada halaman ini bisa dilihat nama aplikasi, port yang digunakan, domain dari aplikasi, dan versi dari aplikasi. Pada halaman ini, pengguna bisa mengatur port dari aplikasi agar dapat berjalan dengan baik. Dari halaman ini juga aplikasi pertama kali akan dijalankan. Jadi pada halaman ini terdapat kontrol untuk menjalankan dan mematikan aplikasi. Antar muka informasi ditunjukkan pada Gambar 4.2.

4.7.3 Daftar Container

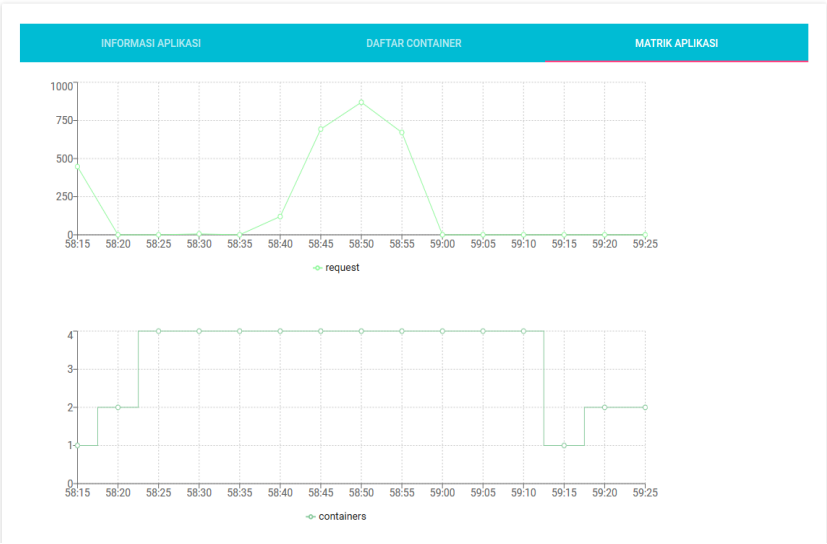
Pada halaman ini, pengguna dapat melihat daftar *container* yang sedang berjalan untuk sebuah aplikasi. Informasi yang diberikan berupa ID dan port dari container. Antar muka halaman daftar *container* ditunjukkan pada Gambar 4.3.

INFORMASI APLIKASI		DAFTAR CONTAINER		MATRIK APLIKASI
<input type="checkbox"/>	No	Container ID	Port	Status
<input type="checkbox"/>	1	5201b6d850	14049	-

Gambar 4.3: Dasbor Daftar Container

4.7.4 Metrik Aplikasi

Halaman metrik aplikasi digunakan untuk memantau keadaan sekarang dari aplikasi. Metrik yang diberikan adalah jumlah *request* pengguna ke aplikasi dan jumlah *container* dari aplikasi. Data akan diperbarui sekitar lima detik sekali. Antar muka metrik aplikasi ditunjukkan pada Gambar 4.4.



Gambar 4.4: Dasbor Matrik Aplikasi

BAB V

PENGUJIAN DAN EVALUASI

5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari: satu *server load balancer*, satu *server master host*, satu *server controller*, satu *server docker registry*, dan enam komputer penguji. Semua *server* menggunakan Virtual Private Server dari DigitalOcean. Lalu, untuk komputer penguji menggunakan lima buah desktop dan satu buah VPS sebagai *docker* klien yang digunakan untuk membuat *docker image*. Pengujian dilakukan di Laboratorium Pemrograman Jurusan Teknik Informatika ITS.

Spesifikasi untuk setiap komponen yang digunakan ditunjukkan pada Tabel 5.1.

Tabel 5.1: Spesifikasi Komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
1	Load balancer	2 core processor, 4GB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, HAProxy, Python 2.7
2	Master host	8 core processor, 16GB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, Docker 17.03.0-ce, Python 2.7
3	Controller	2 core processor, 4GB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, Redis, MySQL, Python 2.7
4	Docker registry	1 core processor, 512MB RAM, 20GB SSD	Ubuntu 14.04.5 LTS, Docker 17.03.0-ce, Python 2.7
5	Komputer penguji	Processor Core2Duo E7300, 2GB RAM	Windows 8, JMeter 3.2

Tabel 5.1: Spesifikasi Komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
6	Docker klien	1 core processor, 1GB RAM, 20GB SSD	Ubuntu 16.04 LTS, Docker 17.03.0-ce

Untuk akses ke masing-masing komponen, digunakan IP publik yang disediakan untuk masing-masing komponen tersebut. Selain menggunakan IP, ada sebagian *server* yang bisa diakses melalui domain. Detailnya ditunjukkan pada Tabel 5.2.

Tabel 5.2: IP dan Domain Server

No	Server	IP dan Domain
1	Load balancer	128.199.160.188
2	Master host	128.199.182.29
3	Controller	128.199.250.137 http://controller.nota-no.life
4	Docker registry	139.59.97.244 https://registry.nota-no.life

5.2 Skenario Uji Coba

Uji coba akan dilakukan untuk mengetahui keberhasilan sistem yang telah dibangun. Skenario pengujian dibedakan menjadi 2 bagian, yaitu:

- **Uji Fungsionalitas**

Pengujian ini didasarkan pada fungsionalitas yang disajikan sistem.

- **Uji Performa**

Pengujian ini untuk menguji ketahanan sistem terhadap sejumlah permintaan ke aplikasi secara bersamaan. Pengujian dilakukan dengan melakukan *benchmark* pada

sistem.

5.2.1 Skenario Uji Coba Fungsionalitas

Uji fungsionalitas dibagi menjadi 2, yaitu uji mengelola aplikasi berbasis *docker* dan uji fungsionalitas menu aplikasi Dasbor.

5.2.1.1 Uji Mengelola Aplikasi Berbasis Docker

Pengujian ini dilakukan untuk mengetahui apakah sistem sudah bisa menyimpan dan mengelola data *docker image* dari aplikasi yang dimasukkan oleh pengembang. Pengujian menggunakan VPS yang berperan sebagai *docker* klien. Pengujian dilakukan dengan memasukkan data *docker image* ke *server docker registry* yang sudah disediakan. Dengan menggunakan sebuah komputer lain yang digunakan untuk membuat aplikasi web berbasis *docker*, dari sana aplikasi tersebut akan ditaruh ke *server docker registry*.

Alamat dari *Docker registry* yang digunakan adalah <https://registry.nota-no.life>. Setelah berhasil melakukan login pada *server docker registry*, selanjutnya adalah memasukkan *image* baru tersebut. *Image* yang dibuat adalah sebuah aplikasi web berbasis PHP 7 dengan *web server* Apache. Aplikasi web tersebut menyediakan sebuah halaman yang berisi sebuah teks yang dibuat melalui pemanggilan fungsi PHP.

Pertama kali *image* tersebut dimasukkan ke *server docker registry*, kemudian data dari *image* tersebut akan disimpan di *server controller*. Setelah data tersimpan, selanjutnya adalah menjalankan aplikasi melalui dasbor yang disediakan. Sebelum menjalankan aplikasi, terlebih dahulu mengatur *port* dari aplikasi yang akan berjalan. Setelah mengatur *port* dengan benar, selanjutnya adalah menjalankan aplikasinya. Jika aplikasi berhasil dijalankan, maka aplikasi dapat diakses melalui domain

yang disediakan.

Setelah aplikasi berjalan, selanjutnya adalah memperbarui aplikasi dengan mengganti teks yang ditampilkan. Untuk itu, pada komputer yang digunakan untuk membuat *image* sebelumnya, maka dibuatkan *image* baru dari aplikasi dengan versi terbaru. *Image* versi terbaru ini kemudian di *push* ke *docker registry*. Setelah selesai melakukan *push*, harapannya aplikasi yang sebelumnya sudah berjalan, akan diperbarui secara otomatis oleh sistem.

Terakhir, pengujian yang dilakukan adalah menghentikan aplikasi yang sudah berjalan. Fungsi untuk menghentikan aplikasi yang sedang berjalan ini terdapat pada dasbor. Jika proses ini berhasil, maka domain yang sebelumnya digunakan untuk mengakses aplikasi akan hilang dan pengguna tidak bisa lagi melakukan akses terhadap aplikasi.

Daftar uji fungsionalitas menambahkan dan memperbarui aplikasi dijelaskan pada Tabel 5.3.

Tabel 5.3: Skenario Uji Mengelola Aplikasi Berbasis Docker

No	Uji Coba	Hasil Harapan
1	Pengguna melakukan <i>login</i> ke <i>server docker registry</i>	Pengguna berhasil melakukan <i>login</i> dengan menggunakan <i>username</i> dan <i>password</i> yang sudah ditentukan.
2	Pengguna menambahkan <i>image</i> baru dari sebuah aplikasi ke <i>server docker registry</i> .	Pengguna berhasil menambahkan <i>image</i> baru dan data tersimpan pada <i>server controller</i> .

Tabel 5.3: Skenario Uji Mengelola Aplikasi Berbasis Docker

No	Uji Coba	Hasil Harapan
3	Pengguna bisa mengatur <i>port</i> dari aplikasi menggunakan dasbor yang disediakan	Data <i>port</i> dari aplikasi yang tersimpan bisa diganti sesuai dengan kebutuhan pengguna.
4	Pengguna bisa menjalankan aplikasi melalui fitur yang ada pada dasbor	Aplikasi berhasil berjalan dan pengguna mendapatkan domain yang digunakan untuk mengakses aplikasi.
5	Pengguna memperbarui aplikasi yang sedang berjalan dengan melakukan <i>push</i> ke <i>server docker registry</i> .	Aplikasi yang sedang berjalan akan diperbarui secara otomatis tanpa perlu perintah dari pengguna.
6	Pengguna menghentikan aplikasi yang sedang berjalan.	Aplikasi berhasil dihentikan dan pengguna tidak bisa lagi melakukan akses aplikasi.

5.2.1.2 Uji Fungsionalitas Menu Aplikasi Dasbor

Aplikasi Dasbor digunakan untuk mengelola dan memantau aplikasi. Aplikasi Dasbor terdiri dari 4 bagian utama, yaitu halaman beranda, informasi aplikasi, informasi *container*, dan metrik dari aplikasi. Rancangan pengujian dan hasil yang diharapkan ditunjukkan dengan Tabel 5.4.

Tabel 5.4: Skenario Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil Harapan
1	Kelola aplikasi	Menambahkan aplikasi baru atau memperbarui aplikasi	Dasbor dapat menampilkan daftar aplikasi terbaru yang dimasukkan atau diperbarui oleh pengembang.
		Menjalankan aplikasi yang sudah masuk ke dalam sistem	Aplikasi dapat berjalan dan pengguna mendapatkan domain untuk mengakses aplikasi.
		Menghentikan aplikasi yang sedang berjalan	Aplikasi yang sedang berjalan dapat dihentikan dan pengguna tidak bisa lagi melakukan akses terhadap aplikasi.
		Mengganti <i>port</i> aplikasi agar dapat berjalan dengan baik	Pengguna dapat mengganti <i>port</i> aplikasi agar aplikasi dapat berjalan dengan benar.

Tabel 5.4: Skenario Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil Harapan
2	Lihat informasi aplikasi	Memilih salah satu aplikasi yang ada	Pengguna dapat melihat informasi secara lengkap tentang aplikasi.
3	Lihat informasi <i>container</i>	Memilih salah satu aplikasi yang ada	Pengguna dapat melihat informasi secara lengkap tentang <i>container</i> yang sedang berjalan untuk aplikasi tersebut.
4	Lihat metrik aplikasi	Memilih salah satu aplikasi yang ada	Pengguna dapat melihat grafik penggunaan CPU dan <i>memory</i> dari aplikasi .

5.2.2 Skenario Uji Coba Performa

Uji performa dilakukan dengan menggunakan lima buah desktop untuk melakukan akses secara bersamaan ke aplikasi menggunakan aplikasi JMeter. Desktop akan mencoba mengakses halaman dari aplikasi web yang sudah berjalan, dengan domain aplikasi.nota-no.life. Halaman yang akan diakses berisi sebuah teks yang dihasilkan dari pemanggilan fungsi PHP.

Percobaan dilakukan dengan lima skenario jumlah *concurrent user* yang berbeda, yaitu sebanyak 800, 1600, 2400, 3200, dan 4000 pengguna dalam rentang waktu inisialisasi ± 15 detik. Waktu tersebut menunjukkan masing-masing pengguna akan mengirimkan request selama ± 15 detik, namun tidak termasuk waktu menunggu balasan dari *server*, yang artinya

keseluruhan permintaan tersebut akan lebih dari waktu tersebut dan bergantung pada kemampuan *server* untuk memberikan respon. Pengujian *request* ini bertujuan untuk mengukur kemampuan dari *proactive model*. Untuk masing-masingnya, dicoba sebanyak empat perhitungan *proactive model* yang berbeda menggunakan ARIMA yang berbeda, yaitu ARIMA(1,1,0), ARIMA(2,1,0), ARIMA(3,1,0), ARIMA(4,1,0). *Proactive model* sendiri berguna untuk mengetahui jumlah *request* kedepannya agar sistem bisa menyediakan sumber daya berdasarkan prediksi tersebut.

Selain itu, untuk memperkirakan sumber daya yang dibutuhkan sistem kedepannya, digunakan *reactive model*. *Model* tersebut akan menghitung jumlah *container* yang sumber daya CPU dan *memory*-nya sudah melebihi batas yang ditentukan. Sistem akan membentuk *container* baru berdasarkan perhitungan *reactive model* tersebut jika ada *container* yang penggunaannya sudah melebihi batas atas dan mengurangi *container* jika ada *container* yang tidak digunakan. Percobaan akan dilakukan sebanyak enam kali dan berikutnya akan dijelaskan data apa yang diuji untuk masing-masingnya.

5.2.2.1 Uji Performa Kecepatan Menangani Request

Pengujian dilakukan dengan mengukur jumlah waktu yang diperlukan untuk menyelesaikan *request* yang dilakukan oleh komputer penguji. Waktu yang diukur adalah perbedaan jarak antara *request* pertama dan yang terakhir dilakukan oleh klien yang mendapatkan balasan dari *server*.

5.2.2.2 Uji Performa Penggunaan CPU

Pengujian dilakukan dengan menghitung penggunaan CPU yang terjadi pada *server master host*. Penggunaan CPU di sini

adalah penggunaan dari *container* aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan CPU dari masing-masing *container* selama proses pengujian dilakukan. Nilai yang didapatkan berupa total persen penggunaan CPU oleh *container* dibandingkan dengan keseluruhan kemampuan CPU.

5.2.2.3 Uji Performa Penggunaan *Memory*

Pengujian dilakukan dengan menghitung penggunaan *memory* yang terjadi pada *server master host*. Penggunaan *memory* di sini adalah penggunaan dari *container* aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan *memory* dari masing-masing aplikasi selama proses pengujian dilakukan.

5.2.2.4 Uji Performa Keberhasilan *Request*

Pengujian dilakukan dengan menghitung jumlah *request* yang gagal dilakukan selama skenario dijalankan. Dari semua jumlah *request* yang dikirimkan selama pengujian, akan didapatkan persen *request* yang gagal dilakukan.

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang telah dijelaskan pada subbab 5.2.

5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang dibangun.

5.3.1.1 Uji Mengelola Aplikasi Berbasis Docker

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1 dan pada Tabel 5.3. Hasil pengujian seperti tertera pada Tabel 5.5.

Tabel 5.5: Hasil Uji Coba Mengelola Aplikasi Berbasis Docker

No	Uji Coba	Hasil
1	Pengguna melakukan <i>login</i> ke <i>server docker registry</i>	OK.
2	Pengguna menambahkan <i>image</i> baru dari sebuah aplikasi ke <i>server docker registry</i> .	OK.
3	Pengguna bisa mengatur <i>port</i> dari aplikasi menggunakan dasbor yang disediakan	OK.
4	Pengguna bisa menjalankan aplikasi melalui fitur yang ada pada dasbor.	OK.
5	Pengguna memperbarui aplikasi yang sedang berjalan dengan melakukan <i>push</i> ke <i>server docker registry</i> .	OK.
6	Pengguna menghentikan aplikasi yang sedang berjalan.	OK.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.3, hasil uji coba menunjukkan semua skenario berhasil ditangani.

5.3.1.2 Uji Fungsionalitas Menu Aplikasi Dasbor

Sesuai dengan skenario pengujian yang dilakukan pada aplikasi dasbor. Pengujian dilakukan dengan menguji setiap

menu pada aplikasi dasbor. Hasil uji coba dapat dilihat pada Table 5.6. Semua skenario yang direncanakan berhasil ditangani.

Tabel 5.6: Hasil Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil
1	Kelola aplikasi	Menambahkan aplikasi baru atau memperbarui aplikasi	Dasbor berhasil menampilkan daftar aplikasi terbaru yang dimasukkan atau diperbarui oleh pengembang.
		Menjalankan aplikasi yang sudah masuk ke dalam sistem	Aplikasi berhasil berjalan dan pengguna mendapatkan domain untuk mengakses aplikasi.
		Menghentikan aplikasi yang sedang berjalan	Aplikasi yang sedang berjalan berhasil dihentikan dan pengguna tidak bisa lagi melakukan akses terhadap aplikasi.

Tabel 5.6: Hasil Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil
		Mengganti <i>port</i> aplikasi agar dapat berjalan dengan baik	Pengguna berhasil mengganti <i>port</i> aplikasi agar aplikasi dapat berjalan dengan benar.
2	Lihat informasi aplikasi	Memilih salah satu aplikasi yang ada	Pengguna berhasil melihat informasi secara lengkap tentang aplikasi.
3	Lihat informasi <i>container</i>	Memilih salah satu aplikasi yang ada	Pengguna berhasil melihat informasi secara lengkap tentang <i>container</i> yang sedang berjalan untuk aplikasi tersebut.
4	Lihat metrik aplikasi	Memilih salah satu aplikasi yang ada	Pengguna berhasil melihat grafik penggunaan CPU dan <i>memory</i> dari aplikasi .

5.3.2 Hasil Uji Performa

Seperti yang sudah dijelaskan pada subbab 5.2 pengujian performa dilakukan dengan melakukan akses ke aplikasi dengan sejumlah pengguna secara bersama-sama. Pengujian dilakukan dengan memberikan *request* secara berkelanjutan dengan jumlah pengguna terdiri dari lima bagian, yaitu 800, 1600, 2400, 3200, dan 4000 pengguna. Untuk jumlah *request* yang dihasilkan dari masing-masing pengguna selama rentang waktu request ± 15 detik dapat dilihat pada Tabel 5.7. Jumlah tersebut akan diolah oleh *reactive model*. Lalu jumlah penggunaan CPU dan memory selama menangani *request* tersebut akan digunakan oleh *proactive model* untuk menambahkan atau mengurangi *container* yang ada.

Tabel 5.7: Jumlah *Request* ke Aplikasi

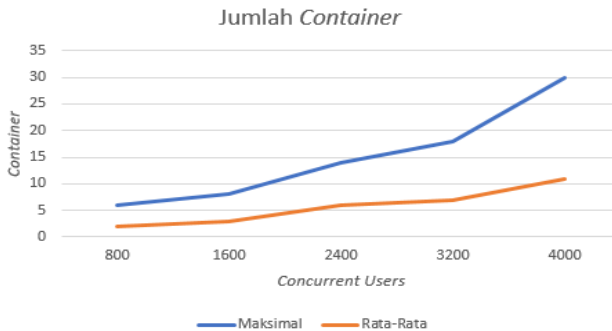
<i>Concurrent Users</i>	<i>Jumlah Request</i>
800	± 16.925
1.600	± 26.650
2.400	± 34.943
3.200	± 50.092
4.000	± 57.750

Pada Tabel 5.8 dapat dilihat jumlah *container* yang terbentuk selama proses *request* dari *user* yang dilakukan selama enam kali. Nilai yang ditampilkan berupa nilai rata-rata selama percobaan dibulatkan ke atas. Sistem dapat menyediakan *container* sesuai dengan jumlah *request* yang diberikan, semakin banyak *request* yang dilakukan, maka *container* yang disediakan akan semakin banyak. Nilai *container* tersebut didapatkan dari perhitungan *proactive model*. Selain melihat jumlah *request*, penentuan *container* yang dibentuk juga dari jumlah sumber daya yang digunakan *container* berdasarkan perhitungan

menggunakan *reactive model*. Pada Gambar 5.1 dapat dilihat grafik dari jumlah *container* yang terbentuk berdasarkan jumlah *request* yang dilakukan.

Tabel 5.8: Jumlah *Container*

<i>Concurrent Users</i>	Maksimal <i>Container</i>	Rata-rata <i>Container</i>
800	6	2
1.600	8	3
2.400	14	6
3.200	18	7
4.000	30	11



Gambar 5.1: Grafik Jumlah *Container*

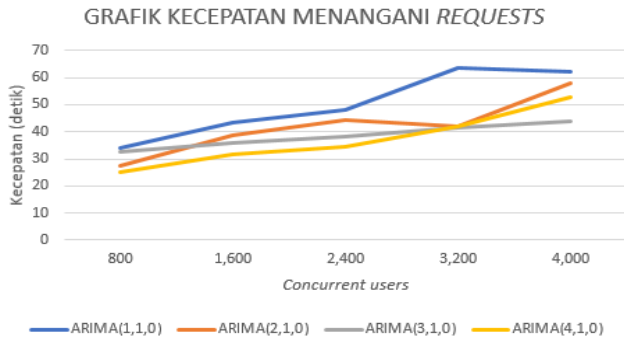
5.3.2.1 Kecepatan Menangani *Request*

Dari hasil uji coba kecepatan menangani *request*, dapat dilihat pada Table 5.9 dalam satuan detik bahwa semakin banyak *concurrent users*, semakin lama pula waktu yang diperlukan untuk menyelesaikannya. Request paling cepat ditangani dengan menggunakan prediksi ARIMA(4,1,0) dan paling lambat

menggunakan ARIMA(1,1,0). Hal tersebut terjadi karena kurang bagusnya hasil prediksi yang dihasilkan oleh ARIMA(1,1,0) yang mana kadang hasil prediksinya terlalu rendah atau terlalu tinggi. Dari hasil percobaan tersebut, dapat dilihat bahwa hampir semua *request* dapat ditangani di bawah satu menit. Lalu grafik hasil uji coba perhitungan kecepatan menangani *request* ditunjukkan pada Gambar 5.2.

Tabel 5.9: Kecepatan Menangani *Request*

	800	1600	2400	3200	4000
ARIMA(1,1,0)	34.167	43.286	48.143	63.857	62.286
ARIMA(2,1,0)	27.429	38.571	44.143	42.143	57.857
ARIMA(3,1,0)	32.429	36.000	38.429	41.571	43.857
ARIMA(4,1,0)	24.857	31.571	34.429	42.143	52.714



Gambar 5.2: Grafik Kecepatan Menangani *Request*

5.3.2.2 Penggunaan CPU

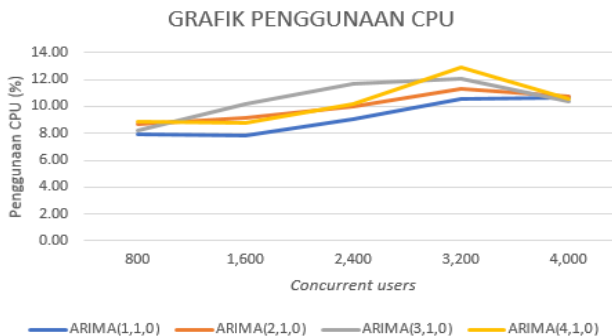
Dari hasil uji coba penggunaan CPU pada *server master host*, penggunaan CPU berada di bawah 15%. Penggunaan CPU yang diukur adalah penggunaan CPU yang dilakukan oleh *container*

dari aplikasi, tidak termasuk sistem. Jumlah *core* yang dimiliki oleh *processor* di *server master host* adalah 8 buah, yang artinya kurang lebih hanya satu *core* yang digunakan untuk menangani semua *request*. Hasil pengukuran penggunaan CPU dapat dilihat pada Tabel 5.10

Tabel 5.10: Penggunaan CPU

	800	1600	2400	3200	4000
ARIMA(1,1,0)	7.1%	7.8%	9.1%	10.5%	10.7%
ARIMA(2,1,0)	8.5%	9.2%	10.1%	11.3%	10.7%
ARIMA(3,1,0)	8.8%	10.2%	11.6%	12.1%	10.3%
ARIMA(4,1,0)	8.0%	8.3%	10.1%	12.9%	10.5%

Dari hasil uji coba, penggunaan prediksi yang berbeda tidak terlalu berpengaruh terhadap penggunaan CPU. Lalu, penggunaan CPU tergolong rendah, yaitu hanya sebesar $\pm 10\%$ untuk menangani semua *request* yang diberikan. Hasil uji coba performa penggunaan CPU ditunjukkan oleh dalam grafik pada Gambar 5.3.



Gambar 5.3: Grafik Penggunaan CPU

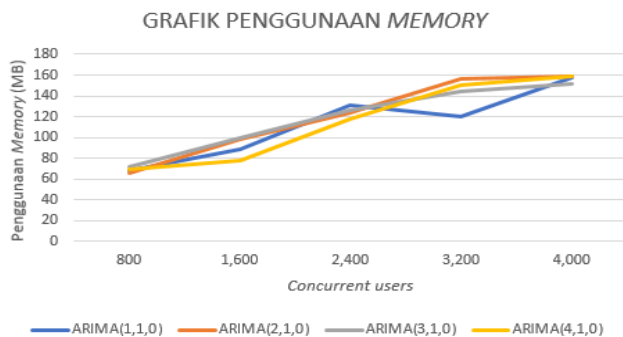
5.3.2.3 Penggunaan *Memory*

Dari hasil uji coba penggunaan *memory*, semakin banyak *request* yang diterima, semakin banyak *memory* yang diperlukan. Perhitungan penggunaan *memory* adalah rata-rata penggunaan dari masing-masing *container* sebuah aplikasi. Untuk masing-masing *container*, dibatasi penggunaan maksimal *memory* adalah 512 MB. Dari hasil uji coba ini, dapat dilihat pada Tabel 5.11 bahwa penggunaan terbesar hanya sebesar 158.71 MB. Artinya jumlah tersebut hanya menggunakan sepertiga dari keseluruhan *memory* yang bisa digunakan.

Tabel 5.11: Penggunaan *Memory*

	800	1600	2400	3200	4000
ARIMA(1,1,0)	67.91	88.97	130.79	120.14	157.73
ARIMA(2,1,0)	65.89	97.98	123.47	156.64	158.33
ARIMA(3,1,0)	72.20	99.72	125.56	144.42	152.14
ARIMA(4,1,0)	69.60	77.34	117.39	149.76	158.71

Hasil uji coba performa penggunaan *memory* dalam grafik ditunjukkan pada Gambar 5.4.



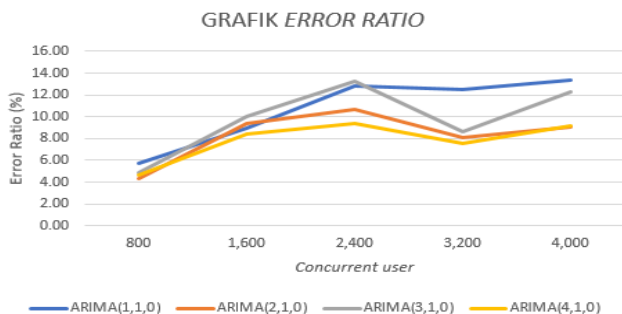
Gambar 5.4: Grafik Penggunaan Memory

5.3.2.4 Keberhasilan *Request*

Pada uji coba ini, dilakukan perhitungan seberapa besar jumlah *request* yang gagal dilakukan. Untuk jumlah *concurrent user* pada tingkat 800 dan 1600, dapat dilihat pada Table 5.12 *error* yang terjadi hampir sama. Prediksi menggunakan ARIMA(4,1,0) berhasil unggul karena menggunakan parameter yang lebih banyak. Namun hal tersebut tidak berlaku untuk ARIMA(3,1,0) karena walaupun parameternya lebih banyak dari ARIMA(2,1,0), tapi hasil prediksinya bisa meleset saat terjadi kondisi dimana koefisien negatif atau koefisien ke dua dikalikan dengan sebuah parameter bukan nol, dan koefisien lain dikalikan dengan parameter nol, maka hasil prediksinya akan negatif, yang mana seharusnya tidak mungkin ada *request* negatif.

Tabel 5.12: *Error Ratio Request*

	800	1600	2400	3200	4000
ARIMA(1,1,0)	5.72%	8.96%	12.85%	12.54%	13.38%
ARIMA(2,1,0)	4.31%	9.35%	10.68%	8.11%	9.04%
ARIMA(3,1,0)	4.84%	10.02%	13.22%	8.63%	12.24%
ARIMA(4,1,0)	4.62%	8.41%	9.39%	7.52%	9.21%



Gambar 5.5: Grafik Error Ratio

Dari uji coba itu, 90% lebih *request* berhasil ditangani. Hasil uji coba jumlah *request* yang gagal ditunjukkan dengan grafik pada Gambar 5.5.

(Halaman ini sengaja dikosongkan)

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 menjalankan dan menyajikan satu atau lebih aplikasi web berbasis *docker* kepada *end-user* melalui domain yang disediakan.
2. Sistem dapat menyesuaikan sumber daya secara otomatis berdasarkan jumlah *request* dengan menggunakan *proactive model* dan penggunaan sumber daya, yaitu CPU dan *memory*, pada *container* dengan menggunakan *reactive model*.
3. Penggunaan *load balancer* cocok digunakan dengan aplikasi yang berjalan di atas *docker container*. Hal tersebut karena semua *request* ke aplikasi akan melalui *load balancer*. Jika terjadi penambahan dan pengurangan sumber daya, penyesuaian dengan cepat dilakukan dan hanya perlu merubah sedikit konfigurasi pada *load balancer* dan pengguna tidak perlu tahu apa yang terjadi di dalam sistem.
4. Prediksi jumlah *request* menggunakan ARIMA sudah bisa menangani skenario uji coba. Perbedaan *order* ARIMA yang digunakan mempengaruhi akurasi dalam menentukan *request* yang akan terjadi ke depannya. Dalam pengujian ini, ARIMA(4,1,0) memiliki hasil pengujian paling bagus dengan jumlah rata-rata *error request* yang paling rendah,

yaitu sebesar 7.83%. Lalu untuk kecepatan menerima *request*, ARIMA(2,1,0) dan ARIMA(4,1,0) memiliki konsistensi yang berbanding lurus dengan jumlah *request*.

5. Penggunaan sumber daya CPU dan *memory* tidak dipengaruhi oleh penggunaan ARIMA yang berbeda. Penggunaan sumber daya tersebut bergantung kepada jumlah *request*, semakin banyak *request* yang diberikan, penggunaan CPU dan *memory* akan semakin tinggi. Penggunaan CPU paling tinggi yaitu sebesar 12.9% dan penggunaan *memory* paling tinggi sebesar 158.71 MB. Dengan penggunaan tersebut, masih tersisa lebih dari setengah sumber daya yang bisa digunakan.
6. Sebuah *container* dari sebuah aplikasi dapat dibentuk dalam waktu ± 1 detik sehingga penambahan sumber daya bisa dilakukan dengan cepat dan proses untuk memperbarui konfigurasi dari HAProxy memerlukan waktu ± 5 detik. Selama proses tersebut, akses pengguna akan tertunda, namun tidak menunjukkan terjadinya *down*.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

1. Mengamankan komunikasi antar *server* karena saat ini *endpoint server* bisa diakses oleh siapapun. Hal tersebut bisa dilakukan dengan mengimplmentasikan *private* IP dan menggunakan token untuk komunikasinya.
2. Pemodelan menggunakan ARIMA cukup baik, namun perlu dicoba untuk melakukan pembuatan model dengan *dataset* yang lebih baru. Selain itu, bisa mencoba alternatif pemodelan *time series* yang lain, seperti ARCH (Autoregressive Conditional Heteroskedasticity).

DAFTAR PUSTAKA

- [1] C. Kan, “DoCloud: An elastic cloud platform for Web applications based on Docker,” in *2016 18th International Conference on Advanced Communication Technology (ICACT)*, Jan. 2016, hal. 478–483.
- [2] “What is Docker?” 2016, 12 Desember 2016. [Daring]. Tersedia pada: <https://www.docker.com/what-docker>. [Diakses: 12 Desember 2016].
- [3] C. Boettiger, “An introduction to Docker for reproducible research, with examples from the R environment,” *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, hal. 71–79, Jan. 2015, arXiv: 1410.0846.
- [4] “1998 World Cup Web Site Access Logs,” 10 April 2017. [Daring]. Tersedia pada: <http://ita.ee.lbl.gov/html/contrib/WorldCup.html>. [Diakses: 10 April 2017].

(Halaman ini sengaja dikosongkan)

LAMPIRAN A

INSTALASI PERANGKAT LUNAK

Instalasi Lingkungan Docker

Proses pemasangan Docker dapat dilakukan sesuai tahap berikut:

- Menambahkan repository Docker

Langkah ini dilakukan untuk menambahkan *repository* Docker ke dalam paket *apt* agar dapat di unduh oleh Ubuntu. Untuk melakukannya, jalankan perintah berikut:

```
sudo apt-get -y install \
    apt-transport-https \
    ca-certificates \
    curl

curl -fsSL https://download.docker.com/linux/
ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/
    linux/ubuntu \
    $ (lsb_release -cs) \
    stable"

sudo apt-get update
```

- Mengunduh Docker

Docker dikembangkan dalam dua versi, yaitu CE (*Community Edition*) dan EE (*Enterprise Edition*). Dalam pengembangan sistem ini, digunakan Docker CE karena merupakan versi Docker yang gratis. Untuk mengunduh Docker CE, jalankan perintah `sudo apt-get -y install docker-ce`.

- Mencoba menjalankan Docker
Untuk melakukan tes apakah Docker sudah terpasang dengan benar, gunakan perintah `sudo docker run hello-world`.

Instalasi Docker Registry

Docker Registry dikembangkan menggunakan Docker Compose. Dengan menggunakan Docker Compose, proses pemasangan Docker Registry menjadi lebih mudah dan fleksibel untuk dikembangkan ditempat lain. Docker Registry akan dijalankan pada satu *container* dan Nginx juga akan dijalankan di satu *container* lain yang berfungsi sebagai perantara komunikasi antara Docker Registry dengan dunia luar. Berikut adalah proses pengembangan Docker Registry yang penulis lakukan:

- Pemasangan Docker Compose

```
$ sudo apt-get -y install python-pip
```

```
$ sudo pip install docker-compose
```
- Pemasangan paket `apache2-utils`
 Pada paket `apache2-utils` terdapat fungsi `htpasswd` yang digunakan untuk membuat *hash password* untuk Nginx. Proses pemasangan paket dapat dilakukan dengan menjalankan perintah `sudo apt-get -y install apache2-utils`.
- Pemasangan dan pengaturan Docker Registry
 Buat folder `docker-registry` dan data dengan menjalankan perintah berikut:

```
$ mkdir /docker-registry && cd $_
```

```
$ mkdir data
```

 Folder `data` digunakan untuk menyimpan data yang dihasilkan dan digunakan oleh *container* Docker Registry. Kemudian di dalam folder `docker-registry` buat sebuah berkas dengan nama `docker-compose.yml` yang akan

digunakan oleh Docker Compose untuk membangun aplikasi. Tambahkan isi berkasnya sesuai dengan Kode Sumber A.1.

```

nginx :
image: "nginx:1.9"
ports :
  - 443:443
  - 80:80
links :
  - registry:registry
volumes :
  - ./nginx/:/etc/nginx/conf.d
registry :
  image: registry:2
  ports :
    - 127.0.0.1:5000:5000
  environment :
    REGISTRY_STORAGE_FILESYSTEM
    _ROOTDIRECTORY: /data
  volumes :
    - ./data:/data
    - ./registry/config.yml:/etc/docker
      /registry/config.yml

```

Kode Sumber A.1: Isi Berkas docker-compose.yml

- Pemasangan *container* Nginx Buat folder nginx di dalam folder docker-registry. Di dalam folder nginx buat berkas dengan nama `registry.conf` yang berfungsi sebagai berkas konfigurasi yang akan digunakan oleh Nginx. Isi berkas sesuai dengan Kode Sumber A.2.

```

upstream docker-registry {
  server registry:5000;
}

```

```

server{
    listen 80;
    server_name registry.nota-no.life;
    return 301 https://
        $server_name$request_uri;
}
server{
    listen 443;
    server_name registry.nota-no.life;
    ssl on;
    ssl_certificate /etc/nginx/conf.d/
        cert.pem;
    ssl_certificate_key /etc/nginx/conf.d
        /privkey.pem;
    client_max_body_size 0;
    chunked_transfer_encoding on;
    location /v2/{
        if ($http_user_agent ~ "^(docker
            \/\1\.(3|4|5(?:!\.[0-9]-dev))|Go )
            .*$" ){
            return 404;
        }
        auth_basic "registry.localhost";
        auth_basic_user_file /etc/nginx/
            conf.d/registry.password;
        add_header 'Docker-Distribution-API
            -Version' 'registry/2.0' always;
        proxy_pass http://docker-registry;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP
            $remote_addr;
        proxy_set_header X-Forwarded-For
            $proxy_add_x_forwarded_for;
    }
}

```

```

    proxy_set_header X-Forwarded-Proto
        $scheme;
    proxy_read_timeout 900;
}
}

```

Kode Sumber A.2: Isi Berkas registry.conf

Instalasi Pustaka Python

Dalam pengembangan sistem ini, digunakan berbagai pustaka pendukung. Pustaka pendukung yang digunakan merupakan pustaka untuk bahasa pemrograman Python. Berikut adalah daftar pustaka yang digunakan dan cara pemasangannya:

- Python Dev
\$ sudo apt-get install python-dev
- Flask
\$ sudo pip install Flask
- docker-py
\$ sudo pip install docker
- MySQLd
\$ sudo apt-get install python-mysqldb
- Redis
\$ sudo pip install redis
- RQ
\$ sudo pip install rq

Instalasi HAProxy

HAProxy dapat dipasang dengan mudah menggunakan apt-get karena perangkat lunak tersebut sudah tersedia pada *repository* Ubuntu. Untuk melakukan pemasangan HAProxy, gunakan perintah apt-get install haproxy.

Setelah HAProxy diunduh, perangkat lunak tersebut belum berjalan karena belum diaktifkan. Untuk mengaktifkan *service haproxy*, buka berkas di `/etc/default/haproxy` kemudian ganti nilai `ENABLED` yang awalnya bernilai `0` menjadi `ENABLED=1`. Setelah itu *service haproxy* dapat dijalankan dengan menggunakan perintah `service haproxy start`. Untuk konfigurasi dari HAProxy nantinya akan diurus oleh *confd*. *confd* akan menyesuaikan konfigurasi dari HAProxy sesuai dengan kebutuhan aplikasi yang tersedia.

Instalasi etcd dan confd

etcd dapat di unggah dengan menjalankan perintah berikut, `curl https://github.com/coreos/etcd/releases/download/v3.2.0-rc.0/etcd-v3.2.0-rc.0-linux-amd64.tar.gz`. Setelah proses unduh berhasil dilakukan, selanjutnya yang dilakukan adalah melakukan ekstrak berkasnya menggunakan perintah `tar -xvzf etcd-v3.2.0-rc.0-linux-amd64.tar.gz`. Berkas binary dari *etcd* bisa ditemukan pada folder `./bin/etcd`. Berkas inilah yang digunakan untuk menjalankan perangkat lunak *etcd*. Untuk menjalankannya, dapat dilakukan dengan menggunakan perintah `etcd --listen-client-urls http://0.0.0.0:5050 --advertise-client-urls http://128.199.250.137:5050`. Perintah tersebut memungkinkan *etcd* diakses oleh *host* lain dengan IP `128.199.250.137`, yang merupakan *host* dari *load balancer* dan *confd*. Setelah proses tersebut, *etcd* sudah siap untuk digunakan.

Setelah *etcd* siap digunakan, selanjutnya adalah memasang *confd*. Untuk menginstall *confd* gunakan rangkaian perintah berikut:

```
$ mkdir -p $GOPATH/src/github.com/kelseyhightower
$ git clone https://github.com/kelseyhightower/
```

```

confd.git $GOPATH/src/github.com/kelseyhightower/
confd
$ cd $GOPATH/src/github.com/kelseyhightower/confd
$ ./build

```

Setelah berhasil memasang confd, selanjutnya buka berkas `/etc/confd/confd.toml` dan isi berkas sesuai dengan Kode Sumber A.3. Pengaturan tersebut bertujuan agar confd melakukan *listen* terhadap server etcd dan melakukan tindakan jika terjadi perubahan pada etcd.

```

confdir = "/etc/confd"
interval = 20
backend = "etcd"
nodes = [
    "http://128.199.250.137:5050"
]
prefix = "/"
scheme = "http"
verbose = true

```

Kode Sumber A.3: Isi Berkas `confd.toml`

Setelah melakukan konfigurasi confd, selanjutnya adalah membuat *template* konfigurasi untuk HAProxy. Buka berkas di `/etc/confd/templates/haproxy.cfg.tmpl`. Jika berkas tidak ada maka buat berkasnya dan isi berkas sesuai dengan Kode Sumber A.4.

```

global
    log /dev/log      local0
    log /dev/log      local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.
        sock mode 660 level admin
    stats timeout 30s

```

```

daemon
defaults
    log      global
    mode     http
    option   httplog
    option   dontlognull
    timeout  connect 5000
    timeout  client  50000
    timeout  server  50000
    errorfile 400 /etc/haproxy/errors
                /400.http
    errorfile 403 /etc/haproxy/errors
                /403.http
    errorfile 408 /etc/haproxy/errors
                /408.http
    errorfile 500 /etc/haproxy/errors
                /500.http
    errorfile 502 /etc/haproxy/errors
                /502.http
    errorfile 503 /etc/haproxy/errors
                /503.http
    errorfile 504 /etc/haproxy/errors
                /504.http
frontend http-in
    bind *:80

    # Define hosts
    {{range gets "/"images/*"}}
    {{$data := json .Value}}
        acl host_{{$data.image_name}}
            hdr(host) -i {{$data.
                domain}}.nota-no.life
    {{end}}

```

```

## Figure out which one to use
{{range gets "/images/*"}}
  {{$data := json . Value}}
    use_backend {{$data .
        image_name}}_cluster if
        host_{{$data . image_name
        }}
    {{end}}
{{range gets "/images/*"}}
  {{$data := json . Value}}
backend {{$data . image_name}}_cluster
mode http
balance roundrobin
option forwardfor
cookie JSESSIONID prefix
{{range $data . containers}}
server {{.name}} {{.ip}}:{{.port}}
    check
  {{end}}
{{end}}

```

Kode Sumber A.4: Isi Berkas haproxy.cfg.tpl

Langkah terakhir adalah membuat berkas konfigurasi untuk HAProxy di `/etc/confd/conf.d/haproxy.toml`. Jika berkas tidak ada, maka buat berkasnya dan isi berkas sesuai dengan Kode Sumber A.5.

```

[ template ]
src = "haproxy.cfg.tpl"
dest = "/etc/haproxy/haproxy.cfg"
keys = [
    "/images"
]

```

```
reload_cmd = "iptables -I INPUT -p tcp --
  dport 80 --syn -j DROP && sleep 1 &&
  service haproxy restart && iptables -D
  INPUT -p tcp --dport 80 --syn -j DROP"
```

Kode Sumber A.5: Isi Berkas haproxy.toml

Setelah melakukan konfigurasi, selanjutnya adalah menjalankan confd dengan menggunakan perintah `confd &`.

Pemasangan Redis

Redis dapat dipasang dengan mempersiapkan kebutuhan pustaka pendukungnya. Pustaka yang digunakan adalah `build-essential` dan `tc18.5`. Untuk melakukan pemasangannya, jalankan perintah berikut:

```
$sudo apt-get install build-essential
$sudo apt-get install tc18.5
```

Setelah itu unduh aplikasi Redis dengan menjalankan perintah

```
wget
http://download.redis.io/releases/redis-
stable.tar.gz.
```

Setelah selesai diunduh, buka file dengan perintah berikut:

```
$tar xzf redis-stable.tar.gz && cd redis-stable
```

Di dalam folder `redis-stable`, bangun Redis dari kode sumber dengan menjalankan perintah `make`. Setelah itu lakukan tes kode sumber dengan menjalankan `make test`. Setelah selesai, pasang Redis dengan menggunakan perintah `sudo make install`. Setelah selesai melakukan pemasangan, Redis dapat diaktifkan dengan menjalankan berkas `bash` dengan nama `install_server.sh`.

Untuk menambah pengaman pada Redis, diatur agar Redis hanya bisa dari `localhost`. Untuk melakukannya, buka file `/etc/redis/6379.conf`, kemudian cari baris `bind`

127.0.0.1. Hapus komen jika sebelumnya baris tersebut dalam keadaan tidak aktif. Jika tidak ditemukan baris dengan isi tersebut, tambahkan pada akhir berkas baris tersebut.

Pemasangan kerangka kerja React

Pada pengembangan sistem ini, penggunaan pustaka React dibangun di atas konfigurasi Create React App. Untuk memasang Create React App, gunakan perintah `npm install -g create-react-app`. Setelah terpasang, untuk membangun aplikasinya jalankan perintah `create-react-app fe-controller`. Setelah proses tersebut, dasar dari aplikasi sudah terbangun dan siap untuk dikembangkan lebih lanjut.

(Halaman ini sengaja dikosongkan)

LAMPIRAN B

KODE SUMBER

Let's Encrypt Cross Signed

-----BEGIN CERTIFICATE-----
MIIEkjCCA3qgAwIBAgIQCgFBQgAAAVOF
c2oLheynCDANBgkqhkiG9w0BAQsFADA/
MSQwIgYDVQQKEExtEaWdpdGFsIFNpZ25h
dHVyZSBUcnVzdCBDby4xFzAVBgNVBAMT
DkRTVCBSb290IENBIFgzMB4XDTE2MDMx
NzE2NDA0NloXDTIxMDMxNzE2NDA0Nlow
SjELMAkGA1UEBhMCVVMxHjAUBgNVBAoT
DUxldCdzIEVuY3J5cHQxIzAhBgNVBAMT
GkxldCdzIEVuY3J5cHQxIzAhBgNVBAMT
IFgzMIIBIjANBgkqhkiG9w0BAQEFAAOA
AQ8AMIIBCgKCAQEAAnNMM8FrILke3cl03
g7NoYzDq1zUmGSXhvb418XCSL7e4S0EF
q6meNQhY7LEqxGiHC6PjdeTm86dicbp5 gWAf15Gan/
PQeGdxyGkOlZHP/uaZ6WA8
SMx+yk13EiSdRxta67nsHjcAHJyse6cF 6
s5K671B5TaYucv9bTyWaN8jKkKQDIZ0
Z8h/pZq4UmEUEz9l6YKH9y9v6Dlb2honz hT+Xhq+
w3Brvaw2VFfn3EK6BlspkENnWA
a6xK8xuQSXgvopZPKiAlKQTGdMDQMc2P
MTiVFrqoM7hD8bEfwbZ/onkxEz0tNvj
/PIzark5McWvxI0NHWWQM6r6hCm21AvA 2
H3DkwIDAQABo4IBfTCCAXkwEgYDVR0T
AQH/BAGwBgEB/wIBADAQBgNVHQ8BAf8E
BAMCAYYwfwYIKwYBBQUHAQEEdBxMDIG
CCsGAQUFBzABhiZodHRwOi8vaXNyZy50
cnVzdGlkLm9jc3AuaWRLbnRydXN0LmNv
bTA7BggrBgEFBQcwAoYvaHR0cDovL2Fw
cHMuaWRLbnRydXN0LmNvbS9yb290cy9k

```

c3Ryb290Y2F4My5wN2MwHwYDVR0jBBgw
  FoAUxKexpHsscfrb4UuQdf/EFWCfiRAw
VAYDVR0gBE0wSzAIBgZngQwBAGewPwYL
  KwYBBAGC3xMBAQEwMDAuBggrBgEFBQcC
ARYiaHR0cDovL2Nwcy5yb290LXgxLmxl
  dHNIbmNyeXB0Lm9yZzA8BgNVHR8ENTAz
MDGgL6AthitodHRwOi8vY3JsLmlkZW50
  cnVzdC5jb20vRFNUUk9PVENBWDNDUkwu
Y3JsMB0GA1UdDgQWBBSoSmpjBH3duubR
  ObemRWXv86jsoTANBgkqhkiG9w0BAQsF
AAOCAQEA3TPXEfNjWDjdGBX7CVW+d1a5
  cEilaUcne8IkCJLxWh9KEik3JHRRHGJo
uM2VcGf196S8TihRzZvoroe6ti6WqEB
  mtzw3Wodatg+VyOeph4EYpr/1wXKtx8/
wApIvJSwtmVi4MFU5aMqrSDE6ea73Mj2
  tcMyo5jMd6jmeWUHK8so/joWUoHOUgwu
X4Po1QYz+3dszkDqMp4fklxBwXRsw10K  XzPMTZ+
  sOPAveyxindmjkW8lGy+QsRlG
PfZ+G6Z6h7mjem0Y+iWlkYcV4PIWL1iw
  Bi8saCbGS5jN2p8M+X+Q7UNKEkROb3N6
KOqkqm57TH2H3eDJAKsnh6/DNFu0Qg==
-----END CERTIFICATE-----

```

Kode Sumber B.1: Let's Encrypt X3 Cross Signed.pem

BIODATA PENULIS



Muhammad Fahrul Razi, akrab dipanggil Razi lahir pada tanggal 23 November 1994 di Ilung, Kalimantan Selatan. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember. Memiliki hobi antara lain membaca novel dan futsal. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Pengembangan Sumber Daya Mahasiswa Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ke-2. Pernah menjadi staff National Programming Contest Schematics tahun 2014 dan 2015. Selain itu penulis pernah menjadi asisten dosen di mata kuliah Pemrograman Jaringan, serta asisten praktikum pada mata kuliah Dasar Pemrograman dan Struktur Data.