BAB III
LANDASAN TEORI

3.1. load balancing

load balancing adalah proses untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara merata antara dua atau lebih computer, link jaringan, CPU, hard drive, atau sumber daya dengan tujuan untuk mendapatkan pemanfaatan sumber daya yang optimal, memaksimalkan throughput, memperkecil waktu tanggap dan menghindari overload. (Abdullah, Simamora, & Andrian, 2010). *load balancing* digunakan pada saat sebuah server telah memiliki jumlah pengguna yang telah melebihi kapasitas maksimal dari server tersebut sesuai dengan kriteria dasar proses *load balancing* mampu mengurangi beban kerja setiap server sehingga tidak ada server yang memiliki kelebihan beban.

3.1.2. Algoritma load balancing

A. Round Robin

Algoritma ini membagi beban secara berurutan dan merata dari satu server ke server lain.

B. Ratio

Algoritma dengan parameter diberikan untuk masing-masing server yang akan dimasukkan kedalam sistem *load balancing*. Server dengan ratio terbesar akan diberi beban besar sedangkan server dengan ratio kecil akan diberi beban kecil.

C.Fastest

Algoritma dengan mengutamakan server-server yang memiliki respon yang paling cepat pada saat permintaan masuk.

D. Least connection

Algoritma membagi beban berdasarkan banyaknya koneksi yang sedang dilayani oleh server. Server yang memiliki koneksi paling sedikit akan melayani permintaan yang masuk.

3.1.3. Tipe *load balancing*

Dalam perancangan *load balancing* terdapat dua pilihan tipe load balancer sebagai berikut:

A. Software *load balancing*

Load balancer yang sering digunakan berbasis perangkat lunak dimana *load balancing* berjalan disebuah server dan aplikasi *load balancing* diinstall dan perlu dikonfigurasi sebelum dapat berfungsi. Proses *load balancing* dipengaruhi oleh perangkat yang digunakan seperti kartu jaringan (Network Interface Card), besarnya RAM (Random Access Memory) dan juga media penyimpanan.

B. Hardware *load balancing*

Load balancer yang berjalan di sebuah alat yang siap digunakan. Tipe *load balancing* ini banyak digunakan karena kemudahannya.

C. Reverse Proxy

Terdapat beberapa jenis sistem penyeimbang beban (*load balancing*) dan salah satu jenisnya merupakan *proxy*. *Proxy* adalah sebuah sistem komputer atau program aplikasi yang melayani permintaan dari klien dengan meminta layanan ke server lain. (Ardhian, dkk,2013).

Proxy server memiliki 3 fungsi utama yaitu :

1. Connection sharing : perantara client dan server.
2. Filtering : bekerja pada layer aplikasi yang dapat memblokir paket-paket tertentu.
3. Caching : mampu menyimpan informasi yang pernah diakses dari server-server.

Proxy di bagi menjadi 2 yaitu *forward proxy* dan *reverse proxy*. *Forward proxy* adalah *proxy* yang meneruskan data ke host tujuan. *Reverse proxy* adalah sebuah *proxy* yang berada di depan dari web server, digunakan sebagai cache atau bisa juga sebagai load balancer. *Reverse proxy* menjadi perantara user-user di internet terhadap akses ke web-web server yang berada pada local area network, sehingga seolah-olah user di internet mengakses langsung web server yang dimaksud padahal sesungguhnya user di internet mengakses web-web server yang terdapat di local area network melalui reverse proxy tersebut.

3.2. Web Server

Web server merupakan perangkat lunak yang melayani permintaan HTTP dari web browser dan mengirimkan kode-kode dinamis ke server aplikasi. Server inilah yang menerjemahkan dan memproses kode-kode dinamis menjadi kode-kode statis dalam suatu halaman statis yang kemudian dikirimkan ke browser oleh web server. Web server biasanya disebut juga dengan HTTP server karena menggunakan protocol HTTP. (Abdullah et al., 2010).

Penggunaan paling umum web server adalah untuk menempatkan situs web. Fungsi utama sebuah web server untuk mentransfer berkas atas permintaan pengguna melalui protokol komunikasi yang telah ditentukan dan mentransfer kedalam sebuah halaman web. Kriteria dasar web server berjalan dengan baik adalah dengan terjalinnya komunikasi misalnya protokol HTTP/HTTPS dari server ke client atau sebaliknya tanpa ada data yang hilang.

3.2.1. Cara Kerja Web Server

Cara kerja dari web server adalah ketika klien meminta halaman web kepada web server, maka permintaan klien tersebut akan membuat sinyal sambungan syncronize dan dikirim ke alamat web server. Selanjutnya web server akan membalas dan mengirimkan sinyal syncronize dan acknowledge kepada klien dan kemudian klien akan membalas dengan sinyal acknowledge yang artinya sudah siap untuk mengirimkan data. Halaman

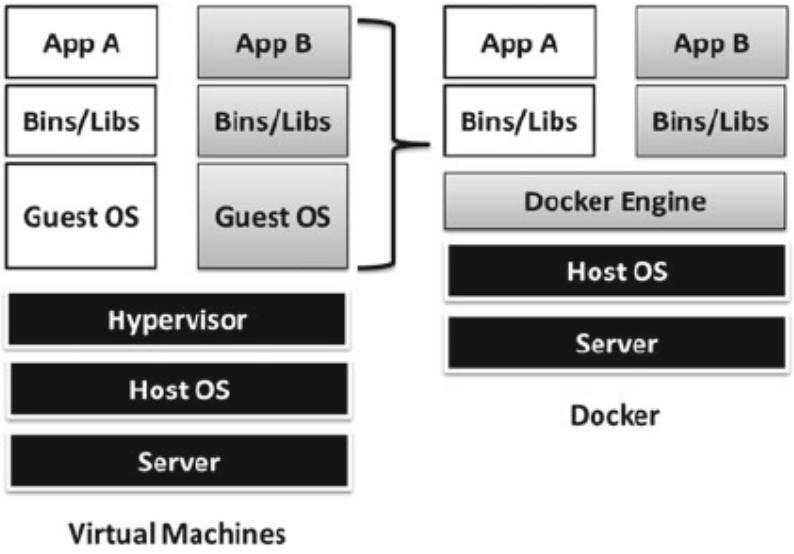
web yang diminta dari klien ke web server disebut dengan HTTP request dan halaman web tersebut dikirim dari web server ke klien yang dikenal dengan HTTP response.

3.2.2. Apache

Apache merupakan program untuk menjalankan,melayani, dan memfungsikan situs web dalam sebuah komputer. Apache dapat dijalankan di banyak sistem operasi (BSD, Linux, Microsoft Windows dan Novell Netware serta platform lainnya) karena merupakan perangkat lunak sumber terbuka dikembangkan oleh komunitas terbuka yang terdiri dari pengembang dibawah naungan Apache Software Foundation. Apache memiliki fitur-fitur canggih seperti pesan kesalahan yang dapat dikonfigurasi, autentikasi berbasis basis data dan lain-lain. Selain itu apache juga bisa digunakan sebagai *load balancer* dengan menambahkan beberapa modul modul konfigurasi tertentu. Pada penelitian ini apache akan di gunakan sebagai *load balancer* adalah apache versi 2.4 karena apache pada versi 2.4 sudah mendukung modul untuk *load balancing* serta space memory yang digunakan lebih sedikit daripada versi sebelumnya (Emanuel, 2015).

3.3. Docker

Docker adalah platform terbuka dimana pengembang dan sistem administrator dapat memanfaatkan untuk membangun, memaketkan, dan *deploy* application, namun tidak ada batasan terhadap mesin server, bahasa pemrograman, atau sistem operasi. Proses *deploy* pada platform *cloud* merupakan salah satu alternatif yang ditawarkan oleh Docker, karena aplikasi dikemas dalam container, sehingga memudahkan untuk ditempatkan di mana saja. Beberapa container dapat berjalan secara bersamaan pada *host* yang sama tanpa intervensi dari *hypervisor*, selain itu setiap aplikasi tersebut dapat berjalan aman terisolasi dalam sebuah container (Martino, Cretella, & Esposito, 2015).



Gambar 3.1. Docker vs Virtual Machine

Gambar 3.1. menunjukkan perbedaan yang ada antara arsitektur menggunakan *hypervisor* untuk menjalankan *Virtual Machine* (di sebelah kiri) dan Docker (di sebelah kanan). Virtual Machine perlu *hypervisors* host pada sistem operasi lokal digunakan untuk menjalankan satu atau lebih guest OS, di mana aplikasi dan *library* berjalan, berbeda dengan Docker dapat langsung diinstal pada OS lokal dan tidak memerlukan perangkat lunak lain untuk membuat *container* bekerja. Secara khusus, Docker terdiri dari dua elemen utama:

1) *Docker Engine*, memilik sifat portabel, *lightweight* runtime dan memaketkan tool, operasi sebagai platform *container virtualization*.
2) *Docker Hub*, sebuah platform Software-as-a-Service untuk berbagi dan mengelola Docker *container* dan aplikasi serta memberikan proses automatisasi workflows. Docker hub bisa diumpamakan sebagai *repository* untuk docker.

Arsitektur Docker didasarkan pada paradigma client-server: *Docker daemon* bertanggung jawab untuk membangun, menjalankan, dan mendistribusikan container Docker, sementara *Docker client* berkomunikasi dengan *docker daemon* untuk mengeluarkan perintah melalui *socket* atau REST API. Pengguna tidak dapat berinteraksi langsung dengan daemon, tetapi mereka harus memanfaatkan fungsionalitas yang telah di

publikasi untuk klien. Client dan daemon dapat dijalankan pada host yang sama atau dapat dihubungi secara remote dari *point of view* pengguna.

Untuk memahami bagaimana Docker bekerja, perlu untuk mengenal elemen internal yang digunakan oleh platform untuk proses *deliver* dan *run container* dan aplikasi didalamnya. Secara khusus, Docker mendefinisikan tiga komponen internal utama:

1. *Docker images* merupakan *read-only* template dari contrainer docker yang telah dijalankan.Terdiri dari lapisan, digabungkan *Union File System* (Unionfs) yang memungkinkan file dan direktori dengan file sistem terpisah membuat lapisan atas transparan. Ketika *Docker image* dimodifikasi, seperti untuk memperbarui aplikasi, layer baru ditambahkan atau yang sudah ada diperbarui: dengan cara ini Docker menghindari seluruh *image* untuk dibangun kembali dan didistribusikan. Setiap *image* dibangun menggunakan image yang telah tersedia, mengandung sistem operasi tertentu, dan dapat dipilih dari *Docker Hub* atau diupload langsung oleh pengguna.

2. *Docker registry* merupakan penyimpanan untuk *Docker image*. penyimpanan tersebut dapat berupa publik atau private. Pengguna umum dapat dicari dari klien dan image yang tersimpan di dalamnya dapat didownload secara bebas dan digunakan. Pengguna private digunakan untuk penelitian dan hanya pengguna terbatas yang dapat mengambil image. *Docker Hub* memberikan dukungan baik untuk publik dan private.

3. *Container* terdiri dari sistem operasi, beberapa file yang ditambahkan pengguna, dan meta-data. *Docker image read-only*, ketika containter menginisiasikan dari docker images, platform otomatis menambahkan *read-write layer* di lapisan atas (menggunakan Unionfs) dimana aplikasi dapat berjalan. Docker memperluas format container yang disebut *Linux Containers* (LXC), dengan *high-level* API menyediakan virtualisasi ringan yang dapat menjalankan proses secara terisolasi untuk menyediakan format container internal. Dalam perkembangannya LXC akan didukung dan format lainnya dapat diintegrasikan.

3.4. HAProxy

3.4.1. Pengertian

HAProxy adalah (Tarreau, 2012):

1) Proxy TCP: dapat menerima koneksi TCP dari listening socket, terhubung ke server dan melampirkan soket ini bersama-sama memungkinkan lalu lintas ke mengalir di kedua arah;

2) HTTP reverse-proxy (disebut "pintu gerbang" dalam terminologi HTTP): menyajikan dirinya sebagai server, menerima permintaan HTTP melalui koneksi diterima pada mendengarkan TCP socket, dan melewati permintaan dari koneksi ini ke server menggunakan koneksi yang berbeda.

3) SSL terminator / inisiator / offloader: SSL / TLS dapat digunakan pada koneksi yang datang dari klien, pada koneksi akan ke server,atau bahkan di kedua koneksi.

4) TCP normalizer : karena koneksi lokal diakhiri oleh operasi sistem, tidak ada hubungan antara kedua belah pihak, membatasi abnormal traffice seperti paket yang tidak valid (invalid flags), flag combination, window advertisement, sequence numbers, incomplete connections (SYN floods), atau tidak meneruskan paket ke sisi lain. Ini melindungi tumpukan fragile TCP dari serangan protokol, dan juga memungkinkan untuk mengoptimalkan parameter koneksi dengan klien tanpa untuk mengubah pengaturan TCP stack server.

5) HTTP normalizer : jika dikonfigurasi untuk memproses lalu lintas HTTP, hanya berlaku jika Permintaan lengkap (complete request) dilewatkan. Ini melindungi terhadap serangan berbasis protokol. Selain itu, protocol deviations dapat ditoleransi dengan dalam spesifikasi tetap sehingga tidak menyebabkan masalah pada server (misalnya: header multiple-line).

6) Content-based switch : dapat mempertimbangkan unsur apapun dari permintaan (request) untuk memutuskan bagaimana server memutuskan untuk meneruskan

permintaan atau terhubung. Sehingga dimungkinkan untuk menangani beberapa protokol melalui port yang sama (misalnya: http, https, ssh).

7) Server load balancer: dapat menyeimbangkan request TCP dan HTTP. Dalam mode TCP, keputusan *load balancing* diambil untuk keseluruhan koneksi. Dalam mode HTTP, keputusan diambil per permintaan (request).

8) Traffic regulator: dapat menerapkan beberapa tingkat pembatasan pada berbagai titik, melindungi server terhadap overloading, menyesuaikan prioritas traffic berdasarkan konten, dan memberikan informasi untuk menurunkan lapisan dan luar komponen jaringan dengan menandai paket.

9) Perlindungan terhadap DDoS dengan penyalahgunaan layanan: dapat mempertahankan sejumlah statistik per alamat IP, URL, *session*, dll dan mendeteksi ketika penyalahgunaan adalah terjadi, kemudian mengambil tindakan (memperlambat pelaku, memblokir mereka, mengirim mereka untuk mengisi data usang, dll).

10) Pengamatan untuk pemecahan masalah jaringan: karena ketepatan informasi yang dilaporkan dalam log, sering digunakan untuk memberikan solusi pada beberapa isu terkait jaringan.

11) - HTTP *compression offloader* : digunakan untuk melakukan pemampatan (*compression*) response yang tidak dapat dilakukan oleh server, hal ini dapat mengurangi beban load time pada halaman klien dengan konektivitas buruk atau dengan tingkat *latency* tinggi, seperti jaringan mobile.

3.4.2. Cara HAProxy Berkerja

HAProxy adalah, *event-driven*, mesin *single-threaded* non-blocking menggabungkan kecepatan lapisan I / O dengan *priority-based scheduler*. Dirancang dengan tujuan data forwarding, arsitektur dioptimalkan untuk memindahkan data secepat mungkin dengan operasi paling mungkin. Karena itu menerapkan model berlapis yang menawarkan mekanisme *bypass* pada setiap level lapisan untuk memastikan data tidak mencapai level yang tidak diinginkan. Sebagian besar pengolahan dilakukan di kernel,

dan HAProxy melakukan yang terbaik untuk membantu kernel melakukan pekerjaan secepat mungkin dengan memberikan beberapa petunjuk atau dengan menghindari operasi tertentu ketika memastikan proses bisa secara berkelompok.

Sebuah proses tunggal dapat menjalankan banyak *proxy instances*; konfigurasi besar seperti 300000 *proxy* yang berbeda dalam suatu proses tunggal dapat berjalan baik. Demikian biasanya tidak ada kebutuhan untuk memulai lebih dari satu proses untuk semua instances.

Hal ini dimungkinkan untuk membuat menjalankan HAProxy lebih banyak proses, tetapi ia datang dengan beberapa keterbatasan. Secara umum tidak masuk akal di HTTP close atau TCP mode karena *kernel-side* tidak dapat berkembang dengan baik dengan beberapa operasi seperti connect(). HAProxy dapat berkembang dengan cukup baik untuk mode keep-alive HTTP tapi kinerja yang dapat dicapai dari sebuah proses tunggal umumnya melebihi kebutuhan umum oleh urutan besarnya. Namun hal itu masuk akal bila digunakan sebagai SSL *offloader*, dan fitur ini juga didukung dalam mode multi-proses (Tarreau, 2012).

3.5. Node.js

Node.js adalah teknologi *single-threaded*. Ini berarti bahwa setiap permintaan diproses hanya dalam satu thread. Dalam bahasa pemrograman lain, seperti, Java, web server menginisiasikan thread baru untuk setiap permintaan. Namun, Node.js dimaksudkan untuk menggunakan pengolahan *asynchronous*, dan ada teori yang menyebutkan bahwa *single thread* dapat memberikan kinerja yang baik. Masalah aplikasi *single-threaded* adalah pemblokiran operasi I / O; misalnya, ketika kita perlu membaca file dari hard disk untuk menanggapi klien. Setelah permintaan baru sampai di server, server membuka file dan mulai membaca request permintaan tersebut. Masalah terjadi ketika permintaan lain yang dihasilkan, dan aplikasi masih melakukan proses pada permintaan pertama (Satheesh, D'mello, & Krol, 2015).

Node.js adalah sistem perangkat lunak yang didesain untuk pengembangan aplikasi web. Aplikasi ini ditulis dalam bahasa JavaScript, menggunakan basis event dan

asynchrounous I/O. Tidak seperti kebanyakan bahasa JavaScript yang dijalankan pada peramban, Node.js dieksekusi sebagai aplikasi server. Aplikasi ini terdiri dari V8 JavaScript Engine buatan Google dan beberapa modul bawaan yang terintegrasi.

3.6. Raspberry Pi

Raspberry Pi (juga dikenal sebagai RasPi) adalah sebuah SBC (Single Board Computer) komputer seukuran kartu kredit yang dikembangkan oleh Yayasan Raspberry Pi di Inggris (UK) dengan maksud untuk memicu pengajaran ilmu komputer dasar di sekolah-sekolah. Raspberry Pi diluncurkan pertama kali pada 29 Februari 2012. Raspberry Pi memiliki dua model, model A dan model B. Harga Resmi untuk model A adalah US$ 25 atau sekitar Rp 250.000 dan model B adalah US$ 35 atau sekitar Rp 350.000 (belum termasuk biaya impor dan pajak ke Indonesia). Perbedaan model A dan B terletak pada memory yang digunakan. Model A menggunakan memory 256 MB dan model B 512 MB. Selain itu model B juga sudah dilengkapi dengan ethernet port (kartu jaringan) yang tidak terdapat di model A. Ada beberapa sistem operasi luar biasa yang bisa digunakan di Raspberry pi, yaitu Linux Debian, Arch Linux ARM, Raspbmc, OpenELEC, dan Android (Pi, 2012).