BAB II

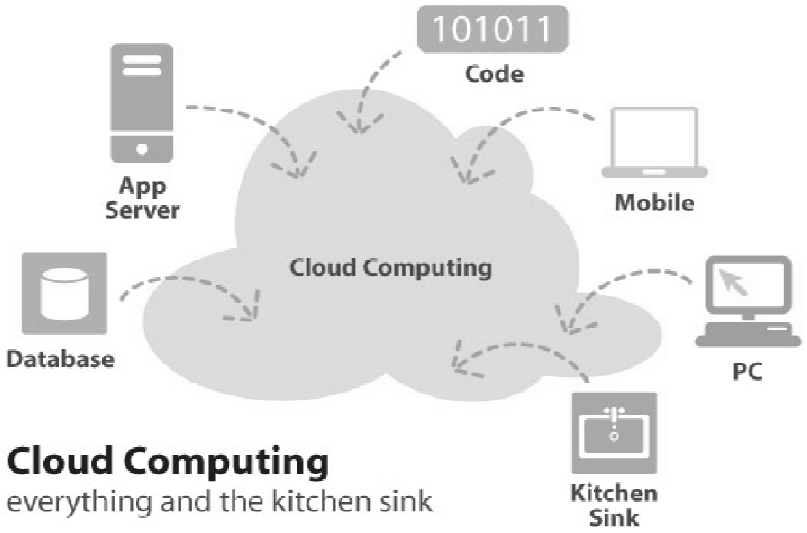
TINJAUAN PUSTAKA

2.1 *Cloud* Computing

2.1.1 Definisi *Cloud* Computing

Cloud Computing terdiri dari dua kata yaitu *Cloud* dan *Computing*. Setiap kata mempunyai arti, *Cloud* bisa dikatakan awan yang dimaksud awan ini adalah internet sedangkan *Computing* adalah proses komputasi.

Cloud Computing adalah evolusi selanjutnya dari internet. *Cloud* pada *Cloud Computing* merupakan penyedia atau hal-hal yang berkaitan dari tenaga komputasi hingga infrastruktur komputasi, aplikasi-aplikasi, proses bisnis hingga kolaborasi yang muncul sebagai layanan yang dapat diakses pada saat dibutuhkan kapanpun dan dimanapun. Teknologi komputasi komputer dengan memanfaatkan internet sebagai terminal utamanya guna mengelola piranti lunak hingga infrastruktur sebagai suatu bentuk layanan.



Gambar 2.1 Ilustrasi *Cloud Computing*

Cloud Computing ditawarkan dalam berbagai bentuk:

1. Public *Cloud*

   Jenis *Cloud* ini diperuntukkan untuk umum oleh penyedia layanannya.

2. Private *Cloud*

   Merupakan infrastruktur layanan *Cloud*, yang dioperasikan hanya untuk sebuah organisasi tertentu. Infrastruktur *Cloud* itu bisa saja dikelola oleh sebuah organisasi itu atau oleh pihak ketiga. Lokasinya pun bisa on-site ataupun off-site. Biasanya organisasi dengan skala besar saja yang mampu memiliki/mengelola private *Cloud* ini.

3. Community *Cloud*

   Dalam model ini, sebuah infrastruktur *Cloud* digunakan bersama-sama oleh beberapa organisasi yang memiliki kesamaan kepentingan, misalnya dari sisi misinya, atau tingkat keamanan yang dibutuhkan, dan lainnya.

4. Hybrid *Cloud*, yang menggabungkan baik public dan private.

   Untuk jenis ini, infrastruktur *Cloud* yang tersedia merupakan komposisi dari dua atau lebih infrastruktur *Cloud* (private, community, atau public). meskipun secara entitas mereka tetap berdiri sendiri, tapi dihubungkan oleh suatu teknologi / mekanisme yang memungkinkan portabilitas data dan aplikasi antar *Cloud* itu. Misalnya, mekanisme loadbalancing yang antar *Cloud*, sehingga alokasi sumberdaya bisa dipertahankan pada level yang optimal

Secara umum *Cloud* ini mirip dengan cairan yang mudah dapat memperluas dan mengisi. Bersifat Elastisitas, yang berarti bahwa pengguna dapat meminta sumber daya tambahan pada permintaannya dan dengan mudah melepaskan

sumber daya ketika mereka tidak lagi diperlukan. Elastisitas ini salah satu alasan utama individu, bisnis, dan pengguna IT berpindah ke awan (*Cloud*). Pada *data center* tradisional itu selalu mungkin untuk menambah dan melepaskan sumber daya.

Untuk evolusi *Cloud Computing* ini sudah berlangsung dan sepenuhnya dapat mengubah cara perusahaan menggunakan teknologi untuk layanan pelanggan, mitra, dan pemasok. Beberapa perusahaan sudah memiliki sumber daya IT hampir seluruhnya di *Cloud*. Mereka merasa bahwa model *Cloud* memberikan keuntungan lebih efisien, biaya- pelayanan IT yang efektif.

Cloud itu sendiri adalah satu set perangkat keras, jaringan, penyimpanan, pelayanan, dan antarmuka yang memungkinkan pengiriman komputasi sebagai layanan. Layanan *Cloud* meliputi pengiriman perangkat lunak, infrastruktur, dan penyimpanan melalui Internet (baik sebagai komponen terpisah atau *platform* lengkap) berdasarkan permintaan pengguna.

2.1.2 Sejarah Perkembangan *Cloud* Computing

Cloud (Awan) adalah suatu istilah yang dipinjam dari telepon. Sampai tahun 1990an, sirkuit data (termasuk yang membawa lalu lintas internet) yang berkabel keras diantara tujuan. Kemudian perusahaan telepon long-haul mulai menawarkan jasa Virtual Private Network (VPN) atau Jaringan Maya Privat untuk komunikasi data. Perusahaan telepon memungkinkan menyediakan layanan yang berdasarkan VPN dengan jaminan bandwidth sebagai sirkuit yang diperbaiki dengan biaya yang lebih murah karena mereka dapat mengganti lalu lintas untuk menyeimbangkan penggunaan yang mereka lihat cocok. Sehingga penggunaan

jaringan mereka secara keseluruhan lebih efektif. Sebagai hasil dari penyusunan ini, memungkinkan untuk menentukan dengan cepat dan tepat jalan mana yang akan dilalui. Simbol *Cloud* (Awan) digunakan untuk menunjukkan tanggung jawab sebuah provider (penyedia layanan), dan *Cloud Computing* (Komputasi awan) memperluasnya untuk melindungi server sebaik infrastruktur jaringannya.

Hal yang mendasari konsep *Cloud Computing* berawal pada tahun 1960-an, saat John McCarthy, pakar komputasi MIT yang dikenal juga sebagai salah satu pionir intelejensi buatan, menyampaikan visi bahwa "suatu hari nanti komputasi akan menjadi infrastruktur publik--sepertilistrik dan telepon". Namun baru di tahun 1995, Larry Ellison, pendiri Oracle, memunculkan ide "Network *Computing*" sebagai kampanye untuk menggugat dominasi Microsoft yang saat itu merajai desktop *Computing* dengan Windows 95-nya. Larry Ellison menawarkan ide bahwa sebetulnya user tidak memerlukan berbagai software, mulai dari Sistem Operasi dan berbagai software lain, dijejalkan ke dalam PC desktop mereka.

PC Desktop bisa digantikan oleh sebuah terminal yang langsung terhubung dengan sebuah server yang menyediakan *environment* yang berisi berbagai kebutuhan software yang siap diakses oleh pengguna. Ide "*Network Computing* " ini sempat menghangat dengan munculnya beberapa pabrikan seperti *Sun Microsystem* dan *Novell Netware* yang menawarkan *Network Computing client* sebagai pengganti desktop.

Namun akhirnya, gaung *Network Computing* ini lenyap dengan sendirinya, terutama disebabkan kualitas jaringan komputer yang saat itu masih belum

memadai, sehingga akses NC (*Network Computing*) ini menjadi sangat lambat, sehingga orang-orang akhirnya kembali memilih kenyamanan PC desktop, seiring dengan semakin murahnya harga PC. Merasakan ketidakpraktisan dengan program-program *web-based*, maka kini diciptakanlah suatu terobosan baru, yaitu *Cloud Computing*. Aplikasi yang ada di *Cloud Computing* tidak tergantung pada sistem operasi yang digunakan oleh pemakai (jadi boleh saja memakai Linux, Mac OS, MS Windows, bahkan sistem operasi PDA atau ponsel).

Yang penting, user dapat mengakses Internet, menuju ke alamat atau situs tertentu, untuk menjalankan program yang dia perlukan. Contoh yang paling mudah dijumpai adalah aplikasi Google (di alamat www.google.com/apps) yang di antaranya terdiri atas organizer (pengelola data relasi, jadwal atau kalender, dan email) dan aplikasi bisnis (pengolah kata, pengolah angka, dan program presentasi). Aplikasi tersebut selain gratis, juga selalu diperbarui oleh pembuatnya. Pemakai tidak perlu membayar apapun, kecuali kalau membutuhkan fitur-fitur yang lebih bagus.

Tonggak selanjutnya adalah kehadiran konsep ASP (*Application Service Provider*) diakhir era 90-an. Seiring dengan semakin meningkatnya kualitas jaringan komputer, memungkinkan akses aplikasi menjadi lebih cepat.

Hal ini ditangkap sebagai peluang oleh sejumlah pemilik *data center* untuk menawarkan fasilitasnya sebagai tempat hosting ‘aplikasi yang dapat diakses oleh pelanggan melalui jaringan komputer. Dengan demikian pelanggan tidak perlu investasi di perangkat *data center*. Hanya saja ASP ini masih bersifat "*private*", di mana layanan hanya dikustomisasi khusus untuks atu pelanggan tertentu,

sementara aplikasi yang di sediakan waktu itu umumnya masih bersifat *client-server*.

Kehadiran berbagai teknik baru dalam pengembangan perangkat lunak di awal abad 21, terutama di area pemrograman berbasis web disertai peningkatan kapasitas jaringan internet,telah menjadikan situs-situs internet bukan lagi berisi sekedar informasi statik. Tapi sudah mulai mengarah ke aplikasi bisnis yang lebih kompleks. Dan seperti sudah sedikit disinggung sebelumnya, popularitas *Cloud Computing* semakin menjulang saat di awal 2000-an, MarcBenioff ex VP di Oracle, meluncurkan layanan aplikasi CRM dalam bentuk *Software as a Service* , Salesforce.com, yang mendapatkan sambutan luar biasa di dunia Teknologi Informasi. Dengan misinya yang terkenal yaitu "*The End of Software*", Benioff bisa dikatakan berhasil mewujudkan visi bos-nya di Oracle, Larry Elisson, tentang *Network Computing* menjadi kenyataan satu dekade kemudian.

Selanjutnya *Cloud Computing* bergulir seperti bola salju menyapu dunia teknologi informasi. Dimulai di tahun 2005, mulai muncul inisiatif yang didorong oleh nama-nama besar seperti Amazon.com yang meluncurkan Amazon EC2 (Elastic Compute *Cloud*), Google dengan Google App Enginenya, tak ketinggalan raksasa biru IBM meluncurkan Blue *Cloud* Initiative dan lain sebagainya. Semua inisiatif ini masih terus bergerak, dan bentuk *Cloud Computing* pun masih terus mencari bentuk terbaiknya, baik dari sisi praktis maupun dari sisi akademis. Bahkan dari sisi akademis, jurnal-jurnal yang membahas tentang hal ini baru bermunculan di tiga tahun belakangan. Akhirnya seperti yang kita saksikan sekarang, seluruh nama-nama besar terlibat dalam pertarungan menguasai awan

ini. Bahkan pabrikan Dell, pernah mencoba mempatenkan istilah "*Cloud Computing*", namun ditolak oleh otoritas paten Amerika. Walaupun di luaran perebutan awan ini begitu dasyat, tidak demikian dengan di tanah air Indonesia tercinta ini. Pemain yang benar-benar mencoba masuk di area ini masih sangat sedikit, bahkan jumlahnya bisa dibilang belum sebanyak jari sebelah tangan. Salah satu yang cukup serius bermain di area ini adalah PT Telkom, yang setidaknya saat ini sudah menawarkan dua layanan aplikasi berbasis *Software as a Service*. Salah satunya melalui anak usahanya, "Sigma Cipta Caraka, yang menawarkan layanan aplikasi core banking bagi bank kecil-menengah. Kemudian bekerjasama dengan IBM Indonesia dan mitra bisnisnya, PT Codephile , Telkom menawarkan layanan e-Office on Demand untuk kebutuhan kolaborasi/korespondensi di dalam suatu perusahaan atau organisasi.

Sepinya sambutan dunia teknologi informasi dalam negeri terhadap *Cloud Computing* ini,mungkin disebabkan beberapa faktor, di antaranya:

1. Penetrasi infrastruktur internet yang bisa dibilang masih terbatas.
2. Tingkat kematangan pengguna internet yang masih menjadikan media interne tutamanya sebagai media hiburan atau sosialisasi.
3. Tingginya investasi yang dibutuhkan menyediakan layanan *Cloud* ini, karena harus merupakan kombinasi antara infrastruktur jaringan, hardware dan software sekaligus.

Namun demikian, sebagai negara dengan jumlah penduduk terbesar ke-5 di dunia, yang berarti juga pasar terbesar ke-5 di dunia, para pelaku teknologi informasi dalam negeri harus sesegera mungkin mempersiapkan diri dalam arti

mulai mengembangkan layanan-layanan yang siap di-*Cloud*-kan. Sehingga saat gelombang besar *Cloud Computing* ini sampai di sini, tidak hanya pemain asing besar saja yang akan menangguk keuntungan. Tentu saja peran pemerintah sebagai fasilitator dan regulator sangat diperlukan di sini. Sampai saat ini paradigma atau pandangan tentang *Cloud Computing* ini masih berevolusi, dan masih menjadi subyek perdebatan yang melibatkan akademisi, vendor teknologi informasi, badan pemerintah, dan pihak-pihak terkait lainnya. Dan untuk memberikan satu common ground bagi publik, pemerintah Amerika melalui National Institut of Science and Technology (NIST) sebagai bagian dari Departemen Perdagangan Amerika, telah membuat beberapa rekomendasi standar tentang berbagai aspek dari *Cloud Computing* untuk dijadikan referensi.

2.1.3 Karakteristik *Cloud Computing*

Sampai saat ini paradigma *Cloud* Computing ini masih berevolusi, masih menjadi subjek perdebatan dikalangan akademisi, vendor IT dan pemeritah/bisnis. Berdasarkan NIST (*National Institute of Standards and Technology*), ada 5 kriteria yang harus dipenuhi oleh sebuah sistem untuk bisa dimasukkan kedalam keluarga *Cloud*

1. *On Demand Self Service*

   Pengguna dapat memesan dan mengelola layanan tanpa interaksi manusia dengan penyedia layanan, Misalnya dengan menggunakan, sebuah portal web dan manajemen portal web dan manajemen antarmuka. Pengadaan dan perlengkapan layanan serta sumber daya yang terkait terjadi secara otomatis pada penyedia.

2. *Broad Network Access*

   Layanan yang tersedia terhubung melalui jaringan pita lebar, terutama untuk dapat diakses secara memadai melalui jaringan internet, baik menggunakan *thin client*, *thick client* ataupun media lain seperti *smartphone*.

3. *Resources Pooling*

   Penyedia layanan *Cloud*, memberikan layanan melalui sumber daya yang dikelompokkan di satu atau berbagai lokasi data *center* yang terdiri dari sejumlah server dengan mekanisme *multi-tenant*. Mekanisme *multi-tenant* ini memungkinkan sejumlah sumber daya komputasi tersebut digunakan secara bersama-sama oleh sejumlah pengguna, di mana sumber daya tersebut baik yang berbentuk fisik maupun virtual, dapat berbentuk fisik maupun virtual, dapat dialokasikan secara dinamis untuk kebutuhan pengguna/pelanggan sesuai permintaan. Dengan demikian, pelanggan tidak perlu tahu bagaimana dan darimana permintaan akan sumber daya komputasinya dipenuhi oleh penyedia layanan. Yang penting, setiap permintaan dapat dipenuhi. Sumber daya komputasi ini meliputi media penyimpanan, *memory*, *processor*, pita jaringan dan mesin virtual.

4. *Rapid elasticity*

   Kapasitas komputasi yang disediakan dapat secara elastis dan cepat disediakan, baik itu dalam bentuk penambahan ataupun pengurangan kapasitas yang diperlukan. Untuk pelanggan sendiri, dengan kemampuan ini seolah-olah kapasitas yang tersedia tak terbatas besarnya, dan dapat "dibeli" kapan saja dengan jumlah berapa saja.

5. *Measured services*

   Sumber daya *Cloud* yang tersedia harus dapat diatur dan di optimasi penggunaannya, dengan suatu sistem pengukuran yang dapat mengukur penggunaan dari setiap sumber daya komputasi yang digunakan (penyimpanan, *memory*, *processor*, lebar pita, aktivitas user, dan lainnya). Dengan demikian, jumlah sumberdaya yang digunakan dapat secara transparan diukur yang akan menjadi dasar bagi user untuk membayar biaya penggunaan layanan.

2.1.4 Model Layanan *Cloud*

Secara umum, layanan pada *Cloud Computing* dibagi menjadi tiga model layanan, yaitu aplikasi/perangkat lunak, *platform* , dan infrastruktur (*Software as a Service, Platform as a Service, dan Infrastructure as a Service*)

1. *Software as a Service* (SaaS)

Pada Software as a *Service*, aplikasi atau perangkat lunak yang dgunakan tersedia di internet. Menyediakan layanan berupa aplikasi yang dapat digunakan oleh penguna yang berjalan pada infrastruktur *Cloud*. Mengapa lebih sederhana bila kita berlangganan perangkat lunak, dibandingkan dengan membelinya?

Karena, masalah maintenance dan support telah diurus oleh *Cloud provider*, dan pengguna hanya menggunakannya.

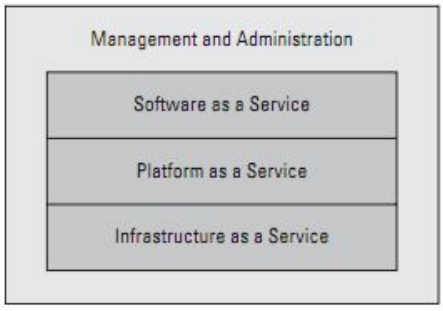
2. *Platform as a Service* (PaaS)

Pada lapis *platform* , kita dapat mendeploy aplikasi yang kita buat di *Cloud*. Menyediakan *platform* (bahasa pemrograman, Tools dsb) guna pengembangan aplikasi berbasis konsumen pada infrasturktur *Cloud.*

Pengguna dapat membuat aplikasi, dan aplikasi tersebut pengguna host di *Cloud provider*. Atau pengguna tidak perlu menginstall perangkat lunak untuk membuat aplikasi, namun dapat mendesain aplikasi, membangun, mendeploy, dan meng-host aplikasi di internet. PaaS juga dikenal dengan *Cloudware.*

3. *Infrastruktur as a Service* (IaaS)

Pada lapis infrastruktur ini, menyediakan sumber daya pemroses, *storage* atau penyimpanan, kapasitas jaringan, dan sumber daya komputasi lainnya. Dimana konsumen dapat mengembangkan dan menjalankan aplikasi khusus seakan-akan mempunyai perangkat keras dan segala isinya pada *remote server*, termasuk perangkat lunak di dalamnya. Maka dari itu, sinonim dari layer ini adalah *Everything as a Service*. Sinonim lainnya adalah *Hardware as a Service*. Secara sederhana, pengguna "menyewa" infrastruktur atau *hardware provider Cloud Computing*, seperti *server space*, *network equipment*, *memory*, CPU *cycle*, dan *storage space*.



Gambar 2.2 Model Layanan *Cloud Computing*

2.1.5 Komponen pada *Cloud Computing*

1. *Cloud Clients*

Cloud Clients adalah seperangkat komputer ataupun perangkat lunak yang didesain secara khusus untuk penggunaan layanan berbasis *Cloud Computing*.

a. Mobile –Windows Mobile, Symbian, dan lain-lain.
b. Thin Client – Windows Terminal *Service*, CherryPal, dll.
c. Thick Client – Internet Explorer, FireFox, Chrome, dll.

2. *Cloud Services*

Cloud Services adalah produk, layanan dan solusi yang dipakai dan disampaikan secara real-time melalui media Internet. Contoh yang paling popular adalah *web service*.

a. Identitas – OpenID, OAuth, dan lain2.
b. Integration – Amazon Simple Queue *Service*.
c. Payments – PayPal, Google Checkout.
d. Mapping – Google Maps, Yahoo! Maps.

3. *Cloud Applications*

Cloud Applications memanfaatkan *Cloud Computing* dalam arsitektur *software*. Sehingga pengguna tidak perlu menginstall dan menjalankan aplikasi dengan menggunakan komputer.

a. Peer-to-peer – BitTorrent, SETI, dan lain-lain.
b. Web Application – Facebook.
c. SaaS – Google Apps, SalesForce.Com, dan lain-lain.

4. *Cloud Platform*

Cloud Platform merupakan layanan berupa *platform* komputasi yang berisi *hardware* dan *software-software* infrasktruktur. Biasanya mempunyai aplikasi bisnis tertentu dan menggunakan layanan PaaS sebagai infrastruktur aplikasi bisnisnya. Contoh :

a. Web Application Frameworks – Python Django, Rubyon Rails, .NET
b. Web Hosting
c. Propietary – Force.Com

5. *Cloud Storage*

Cloud Storage melibatkan proses penyampaian penyimpanan data sebagai sebuah layanan. Misal :

a. Database – Google Big Table, Amazon SimpleDB.
b. Network Attached Storage – Nirvanix *Cloud*NAS, MobileMe iDisk.

6. *Cloud Infrastructure*

Cloud Infrastructure merupakan penyampaian infrastruktur komputasi sebagai sebuah layanan. Contohnya :

a. Grid *Computing* – Sun Grid.
b. Full Virtualization – GoGrid, Skytap.
c. Compute – Amazon Elastic Compute *Cloud*.

2.2 Hosting

Hosting merupakan jasa layanan ruang (space) untuk menyimpan data text, gambar, audio dan video yang selanjutnya file-file tersebut bisa diakses dari mana saja (online). Hosting juga dilengkapi dengan fasilitas protokol seperti FTP dan

HTTP untuk memudahkan dalam mengakses data, selain itu hosting juga bisa digunakan untuk mengelola email.

Biasanya penyedia layanan hosting (hosting provider) menyediakan paket-paket tertentu yang disesuaikan dengan kebutuhan pelanggan, seperti quota space harddisk, quota bandwidth, bahasa pemograman, dan database.

2.2.1 Mengenal beberapa perbedaan jenis hosting

2.2.1.1 *Shared Hosting*

Merupakan layanan hosting yang digunakan bersama-sama oleh pengguna lainnya. Para pengguna layanan ini menggunakan sumber daya baik software maupun hardware atau IP Address secara bersama-sama. Hosting seperti ini dipengaruhi oleh setiap proses yang dilakukan oleh pengguna, jika ada pengguna hosting yang melakukan proses yang berlebihan dan membebani server, maka pengguna lainnya akan terkena imbasnya, seperti server menjadi lambat bahkan ada kemungkinan website menjadi tidak bisa diakses.

Sebagai contoh jika server *Shared hosting* terkena hack, atau hardware failure maka seluruh pengguna *Shared hosting* akan terkena dampaknya seperti website tidak bisa diakses. Pemilik server tentunya akan melakukan pemantauan secara bertahap dan rutin terhadap proses yang terjadi didalam server, biasanya admin akan melakukan suspend account terhadap pengguna yang melakukan pelanggaran atau mengganggu terhadap kinerja server.

Shared hosting cocok bagi Anda yang mempunyai blog, web presence, toko online. Dari sisi harga jenis hosting ini lebih murah jika dibandingkan dengan

jenis hosting lainnya, kembali kepada paket yang disediakan oleh hosting provider.

Kekurangan dari *Shared hosting* ini, pengguna tidak bisa leluasa dalam menggunakan sumber daya server, atau menggunakan software sesuai dengan kebutuhan karena hak penuh terhadap administrasi server dipegang sepenuhnya oleh admin.

2.2.1.2 *Dedicated Hosting*

Jenis hosting ini biasa digunakan untuk solusi hosting terhadap kebutuhan yang berskala besar atau kompleks. Pengguna bisa memilih atau menyediakan spesifikasi server yang diinginkan, yang kemudian ditempatkan pada sebuah data center. Pengguna bisa menggunakan sumber daya server secara leluasa karena administrasi server dikelola atau dimiliki oleh pemilik server.

Dedicated server biasa digunakan oleh penyedia layanan hosting (*Shared hosting* vps), *application service provider*, atau website yang memiliki trafik dan atau membutuhkan resource yang tinggi. Jika dilihat dari harga, pengelolaan dedicated server cukup mahal jika dibandingkan dengan *shared-hosting* atau VPS.

2.2.1.3 *Virtual Private Server*

Virtual Private Server (VPS) merupakan teknologi perangkat lunak yang memungkinkan pembagian sumber daya ke dalam beberapa virtual mesin (server). Tiap-tiap *virtual server* akan mempunyai sistem operasi yang berjalan secara mandiri. VPS menyediakan akses penuh terhadap user root, setiap VPS mempunyai konfigurasi yang bisa diatur layaknya pada server fisik (dedicated server).

Kelebihan dari penggunaan VPS antara lain :

a. Performa yang baik seperti layaknya dedicated server
b. Kebebasan dalam menggunakan sistem operasi atau software yang dibutuhkan
c. Menghemat biaya, jika dibandingkan dengan menggunakan dedicated server

VPS biasanya digunakan pada website atau aplikasi yang memerlukan resource medium diatas shared hosting. Layanan VPS memungkinkan jika pengguna memiliki website atau aplikasi yang memerlukan resource cukup tinggi dengan budget rendah.

2.3 Rails

Ruby on Rails atau sering disingkat Rails merupakan sebuah framework aplikasi web full-stack yang ditulis menggunakan bahasa pemrograman Ruby. Aplikasi web adalah perangkat lunak aplikasi yang diakses menggunakan web browser melalui sebuah jaringan (internet). Sedangkan framework dapat dipandang sebagai fondasi dari aplikasi web yang menangani sebagian besar detail level-bawah yang bersifat repetitive dan membosankan dalam menuliskan kodenya, sehingga pengembang dapat focus pada pembangunan fungsionalitas aplikasi. Full-stack menunjuk kepada luasnya fungsionalitas yang dapat diberikan oleh framework Rails mulai dari penanganan antarmuka pada klien sampai urusan database di server.

Sedangkan Ruby adalah sebuah bahasa scripting berorientasi objek yang diciptakan pertamakali oleh Yukihiro Matsumoto pada awal 1990-an (Lenz,

2008:2). Berikut beberapa prestasi yang telah berhasil diraih oleh framework web ini:

a. Rails memenangkan Jolt Award tahun 2006 sebagai perkakas pengembangan web terbaik.
b. Buku "Agile Web Development with Rails" terbitan The Pragmatic Programmers menerima penghargaan Jolt Award 2006 untuk kategori buku teknik terbaik.
c. Penemu Rails, David Heinemeier Hansson diberi gelar sebagai Hacker of the Year pada OSCON 2006.
d. Banyak tokoh-tokoh programmer yang beralih dari PHP dan Java ke Rails seperti James Duncan Davidson (pengembang Tomcat dan Ant) dan Bruce Tat (pengarang buku-buku Java).
e. Rails telah menjadi salah satu framework yang paling berpengaruh dala pengembangan aplikasi web dengan banyaknya framework web lain yang menggunakan filosofi dan arsitektur Rails, seperti Merb, Django, Grails, CakePHP, Code Igniter, JBOSS Seam, ASP.NET dll.

2.3.1 Sejarah Ruby on Rails

Ruby on Rails berasal dari ekstraksi sebuah aplikasi manajemen proyek, Basecamp, yang dikembangkan oleh pengembang web dari Denmark, David Heinemeier Hansson, untuk perusahaan 37signals. David merilis Rails ke publik untuk pertamakalinya pada Juli 2004 sebagai perangkat lunak open source, dan dilanjutkan dengan rilis versi 1.0 pada 13 Desember 2005. Sekarang Rails telah mencapai Versi yang dirilis pada November 2008 dan di-maintenance oleh Rails

Core Team dibantu dengan patch dan bugfix dari jutaan komunitas pengguna Rails. (Lenz,2008:4)

2.3.2 Prinsip-prinsip Pengembangan

Menurut Lenz(2008:5), Rails mendukung beberapa prinsip pengembangan perangkat lunak yang membuatnya dapat terus bertahan diantara framework pengembangan web lainnya. Dengan adanya prinsip-prinsip ini maka Ruby on Rails betul-betul mengurangi waktu dan usaha yang digunakan oleh pengembang dalam membangun aplikasi web yang kompleks.

2.3.2.1 Convention over Configuration (CoC)

KOnsep ini merujuk kepada fakta bahwa Rails mengasumsikan sejumlah cara default yang harus dilakukan oleh seseorang ketika membangun aplikasi web sederhana. Ini berbeda dengan kebanyakan framework lain seperti framework web Struts (Java) dan Zope (PHP) yang membutuhkan proses konfigurasi yang panjang sebelum pengembang dapat memulai mengembangkan aplikasi paling sederhana sekalipun. Informasi konfigurasi ini biasanya disimpan dalam sebuah file XML, dan file dapat menjadi sangat besar dan akan menjadi sangat rumit untuk memeliharanya. Selain itu, pengembang mesti mengulangi seluruh proses konfigurasi setiap akan memulai proyek yang baru.

Sementara itu, Rails dikembangkan bertujuan untuk menghilangkan konfigurasi yang berlebihan dan digantikan dengan adanya konvensi standar. Hasilnya, tidak dibutuhkan lagi file konfigurasi yang panjang. Jika pengembang mengikuti konvensi standar ini secara default, maka ia hanya perlu berhubungan

dengan satu file konfigurasi yang singkat saja yang digunakan untuk membangun koneksi database.

Konvensi lain yang disediakan oleh Rails, diantaranya penamaan item-item yang berkaitan dengan database dan proses-proses yang melibatkan controller menemukan model dan view yang berkaitan

2.3.2.2 Don’t Repeat Yourself (DRY)

Rails mendukung prinsip pemrograman DRY (Don’t Repeat Yourself). Ketika pengembang memutuskan untuk mengubah kelakuan dari aplikasi yang menggunakanprinsip DRY, maka ia tidak perlu untuk memodifikasi kode aplikasi pada lebih dari satu lokasi.

2.3.2.3 Pengambangan Agile

Banyak pendekatan tradisional dalam pengembangan perangkat lunak seperti iterative development dan waterfall model, biasanya mencoba untuk menguraikan secara detail dalam sebuah rencana statis yang panjang dengan menggunakan metoda prediktif utuk menentukan tujuan dan kebutuhan aplikasi.

Berlawanan dengan itu, metode pengembangan agile menggunakan pendekatan adaptive. Suatu tim yang kecil (biasanya kurang dari 10 pengembang)secara iteratif menyelesaikan suatu unit kecil dari proyek. Sebelum memulai sebuah iterasi, tim mengevaluasi ulang prioritas untuk aplikasi yang sedang dibangun (prioritas-prioritas ini mungkin telah berubah ketika iterasi sebelumnya), sehingga tim mungkin perlu melakukan penyesuaian. Pengembang agile juga mengembangkan aplikasinya secara top-down, dimulai dari desain (yang mungkin saja sederhana berupa sketsa antarmuka pada selembar kertas).

Ketika sebuah aplikasi dibangun menggunakan metode agile, mungkin saja terjadi pembelokan kontrol disepanjang siklus pengembangan, dikarenakan upaya yang terus-menerus yang dilakukan tim untuk memperbaiki prioritas. Dengan menghabiskan waktu yang sedikit pada pembuatan spesifikasi fungsional dan jadwal long-running, pengembang yang menggunakan metodologi agile dapat sesegera mungkin memulai sebuah pengembangan aplikasi.

Berikut ini sedikit contoh yang mengilustrasikan bagaimana Rails mendukung praktik pengembangan agile:

a. Pengembang dapat memulai bekerja pada layout dari aplikasi Rails sebelum membuat keputusan apapun terhadap penyimpanan data (walaupun keputusan- keputusan ini mungkin berubah pada tahap selanjutnya) dan ia tidak perlu mengulangi mengerjai layout ini ketika memulai memasukkan fungsionalitas pada layar desain, segalanya berkembang secara dinamis seiring dengan persyaratannya.
b. Tidak seperti kode yang ditulis menggunakan C atau Java, aplikasi Rails tidak memerlukan sebuah langkah kompilasi supaya bisa menjadi executable. Kode Ruby diinterpretasikan secara on-the-fly, sehingga ia tidak membutuhkan bentuk kompilasi binari apapun untuk membuatnya executable. Perubahan kode di sepanjang pengembangan akan tetap memberikan pengembang feedback yang sesegera mungkin, yang dapat mempercepat laju pengembangan aplikasi secara signifikan.

2.3.3 Karakteristik Rails

Rails 2 mempunyai karakteristik yang berbeda dibandingkan web framework lainnya diantaranya :

1. Berbasis arsitektur MVC (Model-View-Controller)

   Pemisahan data, presentasi dan logika kontrol dalam 3 tempat yang terpisah yang lebih memudahkan dalam memelihara kode yang sedang dibuat dan mengontrol perpindahan data.

   a. Model

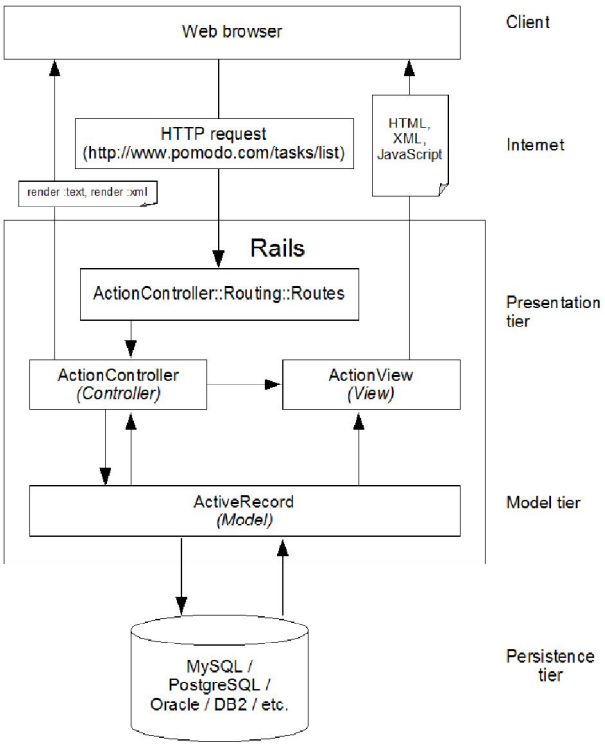
      Arsitektur model diatur oleh ActiveRecord yang merupakan salah satu lapisan dalam kode Rails yang menyediakan ORM (Object-Relational Mapping) antara Rails dan database.

   b. View

      Bagian view dari arsitektur menangani presentasi data, misalnya pada sebuah web browser.

   c. Controller

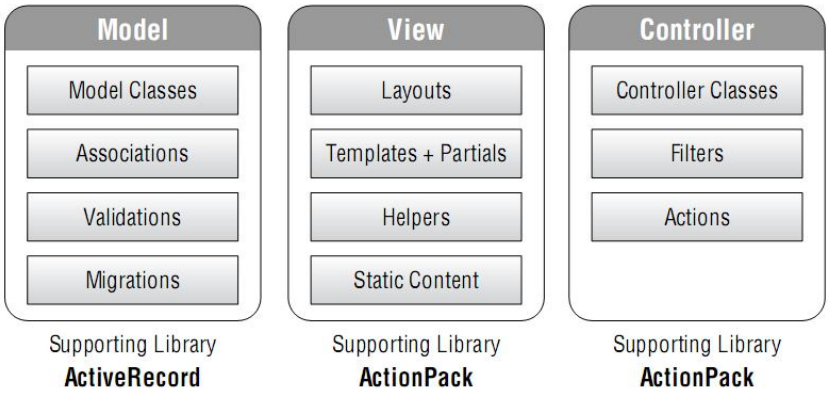
      Controller bersama-sama dengan view tergabung dalam ActionPack, bertugas mengambil masukkan pengguna dan merespon dengan meminta suatu operasi pada model dan mempersiapkan hasilnya untuk ditampilkan oleh bagian view.



Gambar 2.3 Arsitektur MVC pada Rails (Armstrong, 2008:6)

2. Rails secara *default* mengimplementasikan arsitektur REST.

3. Rails merupakan sebuah *framework* yang *full-stack.*



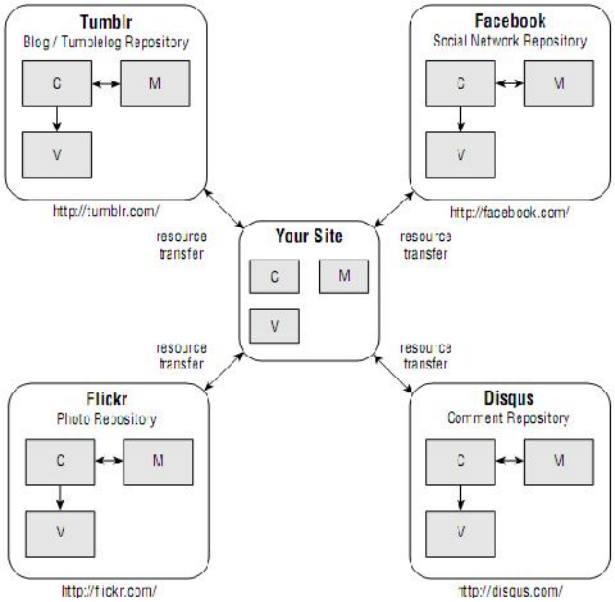
Gambar 2.4 Kelengkapan Rails dengan Berbagai Pustaka (Benson, 2008:25)

4. *Script.* Banyak *script* Rails tersedia yang memudahkan otomatisasi tugas-tugas pada siklus pengembangan yang dapat mengurangi beban kerja.

5. Validasi. Rails memiliki metoda untuk memvalidasi apapun mulai dari pembuatan, penampilan, ukuran atau panjang sesuatu dan lain-lain.
6. AJAX. Rails mendukung AJAX dimana sebuah browser dapat meng-*update* atau mengubah sebagian dari halaman web yang merupakan objek XMLHTTPRequest sehingga memungkinkan *update* dilakukan tanpa sepengetahuan dan persetujuan dari pengguna browser (*backgrounding*).
7. *Migration.* Rails memiliki kemampuan untuk melakukan migrasi dari satu skema *database* ke yang lainnya dengan mudah hanya dengan melalui kode Ruby saja.
8. *Console.* Rails mengijinkan pengembang untuk menguji aplikasi Rails secara *black-box* menggunakan perangkat **irb** dalam sebuah konsol.
9. *Environment* dan Pengujian. Secara *default*, Rails menyediakan tiga *environment* paralel: *development*, *test* dan *production*. Demikian juga Rails menyediakan dukungan yang lengkap terhadap aktivitas pengujian.
10. Rake. Perkakas pengembangan mirip *make* yang ditulis dalam bahasa Ruby yang berguna dalam menyelesaikan tugas-tugas yang berkaitan dengan proyek seperti pembuatan/penghapusan *database* maupun pengubahan skema *database* atau pemrosesan file-file dalam jumlah banyak.

2.4 REST (Representational State Transfer)

REST mempersatukan teori tentang bagaimana "*distributed hypermedia system*" (terutama *World Wide Web*) diorganisir dan distrukturkan dengan sebaik mungkin. REST merupakan cara baru berpikir tentang arsitektur jaringan berdasarkan pengamatan atas bagaimana jaringan bekerja.



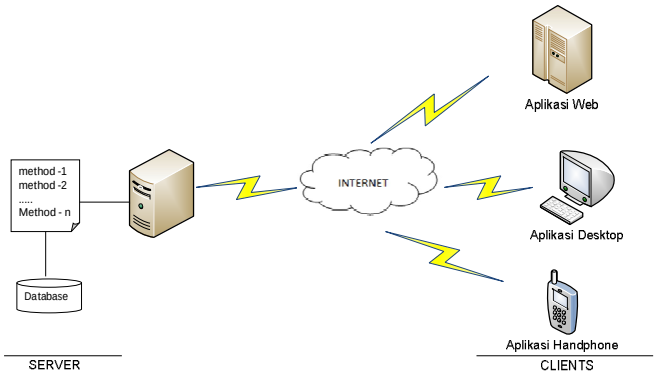
Gambar 2.5 Web dengan Gaya Arsitektur REST (Benson, 2008:128)

2.5 *Web Service*

Web service adalah kumpulan dari fungsi atau *method* yang terdapat pada sebuah *server* yang dapat dipanggil oleh klien dari jarak jauh. Untuk memanggil *method -method* tersebut, kita bebas menggunakan aplikasi yang dibuat dengan bahasa pemrograman apa saja yang dijalankan diatas *platform* apa saja. Inilah keunggulan utama dari *web service*.

Pada masa sekarang ini perangkat keras, sistem operasi, aplikasi, hingga bahasa pemrograman semakin beraneka ragam jenisnya. Keadaan tersebut dapat menimbulkan masalah dalam proses pertukaran data antar perangkat yang menggunakan aplikasi dan *platform* yang berbeda.

Untuk memecahkan permasalahan tersebut, diciptakanlah *web service*. Penggunaan *web service* memugkinkan perangkat-perangkat yang menggunakan sistem operasi dan aplikasi yang berbeda satu sama lain dapat saling bertukar data dan informasi dengan mudah.



Gambar 2.6 *Web Service* Dapat Diakses Berbagai Jenis Aplikasi

Penyediaan *service* pada aplikasi web yang dibuat menjadi penting dalam mencapai kesuksesan. Ini memberikan pengguna kontrol yang lebih baik terhadap data yang disediakan situs dan membuka pintu kreatifitas guna-ulang dari data tersebut. *Web service* juga mempromosikan sebuah visi dari Web dimana situs mampu menspesialisasikan fokusnya dan bekerjasama dengan lebih efisien dalam mencapai tujuan pemrograman. Contohnya, sekarang pengembang Web lebih cenderung menggunakan salah satu peta (*map*) yang ditawarkan Google, Yahoo! Atau Microsoft ketimbang membuat sendiri petanya. Dalam area teknologi manapun, guna-ulang komponen dan kemampuan untuk spesialisasi merupakan faktor kunci dalam meraih kemajuan. Penggunaan dan pengembangan *service* pada Web memungkinkan peningkatan kualitas dan pengalaman aplikasi web untuk semua pengguna.

Rails dalam hal ini telah menyediakan gaya yang khas dalam mendesain *service* dimana seorang pengembang dipaksa untuk mendesain *website* dan *web service* sekaligus dalam satu usaha ketimbang sebagai dua hal yang terpisah dan

komponen yang berbeda dari aplikasi web. Untuk itu, terdapat dua ide cemerlang yang akan mengubah cara mendesain dan membuat *web service*.

2.5.1 URL yang Mengalamati Konsep, Bukan File

Dahulu, URL beroperasi pada lapisan abstraksi yang disediakan oleh file system fisik (file dan direktori), tetapi sekarang URL dioperasikan dalam satu lapisan abstraksi baru yang ditambahkan di atas aplikasi dan digunakan sebagai ruang yang dapat dialamati.

Dahulu, walaupun kode aplikasi dan strukturnya berarti bagi pengembang, tapi itu tidak berarti bagi pengguna. Sekarang dengan adanya lapisan abstraksi baru ini, maka URL bisa terlihat "cantik" dan berarti bagi pengguna juga. Ini misalnya terdapat pada Flickr, URL akan berbentuk seperti di bawah ini: **http://flickr.com/photos/deritaf**

URL ini bisa dibandingkan dengan URL pada masa dahulu, misalnya URL berikut ini yang menunjuk pada *merchandise* Dr.Seuss dari situs Amazon pada tanggal 2 Maret 2000 yang didapatkan dari Internet Archive:

http://s1.amazon.com/exec/varzea/search-handle-rl/ref=gw_m_col_7/?index=fixed-price%26rank=-price%26field-status=open%26field-browse=68457%26field-titledesc%3DDr.%20Seuss.

Konsep baru URL ini dapat dijelaskan sebagai berikut:

a. URL dipandang sebagai alamat menuju ruang-konsep dalam aplikasi ketimbang sebagai alamat menuju kode di dalam aplikasi. URL ini mungkin merepresentasikan noun (resource yang diatur oleh aplikasi) dan verb (action pada resource-resource tersebut). URL merupakan entiti yang berarti walau

tanpa parameter apapun, tetapi parameter tetap dapat digunakan untuk mengklarifikasi dan menambahkan parameter tambahan terhadap request.

b. URL yang dipetakan menuju ruang-konsep aplikasi benar-benar direncanakan (engineered), sebagaimana halnya dengan interface objek pada bahasa pemrograman Java dan C++. Struktur dari URL harus mengikuti pola-pola yang mudah dibaca dan dimengerti. Pola-pola ini dituliskan dan dipaksakan sebagai bagian dari aplikasi web.

c. Kecuali konten statik, maka tidak terdapat hubungan antara URL dengan file di web server. URL mengalamati lokasi dalam ruang-konsep, di filesystem. Suatu langkah yang disebut “routing” akan terjadi manakala sebuah web request diterima yang akan mengambil alamat konseptual ini dan memutuskan bagian mana dari kode aplikasi yang akan digunakan untuk menjawab request tersebut.

2.5.2 Aplikasi Sebagai Service

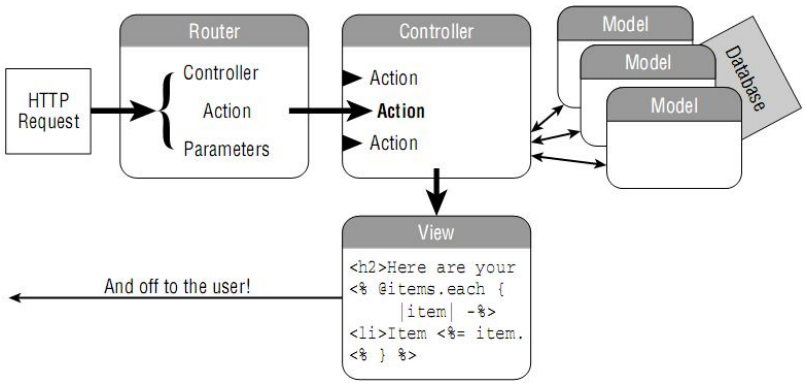
Setiap kali suatu halaman pada situs dimuat, halaman tersebut merupakan merupakan respon terhadap suatu *request service*. Fakta bahwa halaman web tersebut direnderdalam HTML adalah hanya karena itu merupakan tipe respon dari *request service* tersebut. Informasi yang direpresentasikan oleh halaman web tersebut dapat juga direpresentasikan dengan baik dalam bentuk file teks, XML, RDF atau format lainnya yang dipilih.

Jika ide sebelumnya diikuti dimana URL seharusnya merepresentasikan konsep ketimbang file, maka ini berarti pengembang sudah mendefinisikan “antarmuka” bagi *service*, serangkaian konsep yang akan membuat website tersedia bagi penggunanya.

Sekarang tinggal mengimplementaikan *back-end* yang mampu merespon terhadap *request* non-HTML pada *endpoint* URL yang sama.

2.5.3 Routing

Routing memainkan peran kunci yang memungkinkan pengembangan web service ini pada Rails. Routing adalah sebuah proses dimana request HTTP yang datang dipasangkan/dicocokkan dengan suatu bagian tertentu dari kode, yang dalam hal ini merupakan sebuah action pada controller yang akan merespon terhadap request tersebut. Pada proses ini, Rails akan melakukan scan terhadap URL request yang datang untuk dicocokkan dengan serangkaian route yang dispesifikasikan dalam file config/routes.rb di dalam project.



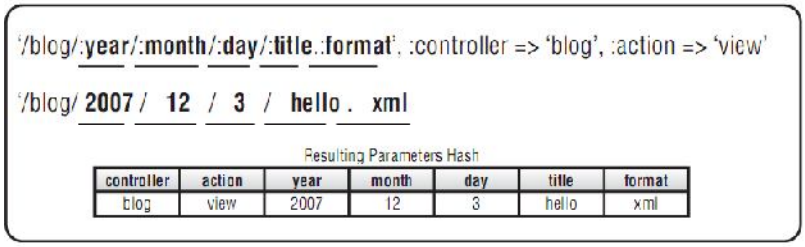
Gambar 2.7 Proses yang Terjadi Ketika Request HTTP Datang

Berikut ini penjelasan bagaimana proses *routing* ini bekerja:

a. Ketika satu *request* baru tiba di server, maka pertama-tama web server akan mengecek apakah file-file statik pada direktori /public merupakan respon yang diinginkan.
b. Jika ya, URL diinterpretasikan sebagai lokasi file dalam *filesystem* lokal dari aplikasi web dan konten file akan dikirimkan ke pengguna.

c. Jika path URL tidak sesuai dengan file yang terdapat pada direktori /public, maka selanjutnya *request* akan ditangani oleh *router* Rails yang akan mencocokkan *request path* dengan *route* yang diketahui.

d. Proses *routing* menggunakan serangkaian definisi *route* pada file config/routes.rb yang dibuat pengembang untuk mendefinisikan *endpoint* URL dimana aplikasi web akan merespon. Setiap route merupakan pola yang akan diisi. Route diperiksa dengan urutan kemunculannya pada file routes.rb dimana yang pertama kali muncul yang cocok dengan *path request* akan menjadi penentu bagaimana *request* yang datang tersebut ditangani.



Gambar 2.8 Contoh Definisi Route

Karena file routes.rb merupakan penghubung yang menjembatani antara semua *request* web non-statik dengan kode aplikasi, maka jelaslah bahwa desain URL adalah langkah yang sangat terencana (*engineered*) dari sejak awal pengembangan aplikasi web dimulai dan merupakan langkah yang eksplisit harus dilakukan pada pengembangan aplikasi Rails. Halaman web manapun harus memiliki sebuah URL yang telah didefinisikan dengan sebuah *route*. *Route-route* ini merepresentasikan antarmuka publik dari sebuah *service*, walaupun jika *service* tersebut hanya mengembalikan halaman HTML.

Dalam sebuah API, metode yang *protected* dan *private* yang menyelesaikan tugas/pekerjaaan level rendah benar-benar tersembunyi dari pengguna API. Demikian

juga dalam *web service* dengan *route*, struktur file aktual aplikasi web yang mengerjakan tugas/pekerjaan benar-benar tersembunyi dari pengguna. Ini merupakan sebuah pemisahan yang lengkap antara file yang menyebabkan aplikasi web berjalan dan pola URL yang menyediakan akses terhadap fungsionalitas tersebut.

2.5.4 Anatomi Pemanggilan *Web Service*

Adanya Routing URL, memungkinkan bagi API berbasis HTTP yang user-friendly dan mengizinkan URL untuk merepresentasikan endpoint virtual di dalam sebuah fungsionalitas aplikasi. Endpoint-endpoint ini saja tidak cukup untuk menspesifikasikan pemanggilan web service secara sempurna melainkan diperlukan empat komponen yang berbeda.

Empat komponen ini, seperti dideskripsikan pada tabel di bawah ini, terlihat hampir mirip dengan komponen-komponen pada pemanggilan metode tradisional, dengan beberapa kekecualian. Fungsi pemanggil dapat meminta tipe pengembalian, dan sebuah perintah HTTP diberikan bersamaan dengan metode pemanggilan sebagai potongan data yang akan memandu eksekusi metode tersebut.

Tabel 2.1Komponen pada Pemanggilan Web Service

Komponen	Disediakan Oleh	Tujuan
Controller dan *Action*	Definisi *route*	Memilih sebuah kelas *controller* dan metode dalam kelas tersebut untuk dieksekusi sebagai respon terhadap *web request*
Format Respon	*Header* HTTP atau definisi *route* (variabel :format)	Menspesifikasikan format respon dari action yang tersedia
Parameter *Request*	*Form* data atau parameter yang di-*encode* pada URL	Memberikan parameter tambahan untuk memenuhi *request* dasar
Perintah HTTP	*Request HTTP*	Menyatakan *request* yang diinginkan, apakah itu mengambil data, memodifikasi data atau menghapus data.

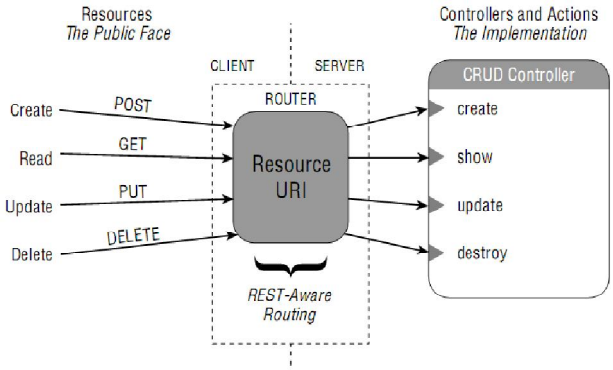
Ketika mendefinisikan route, ini merupakan konteks yang lebih luas dimana mereka akan saling sesuai. Website merupakan koleksi dari endpoint-endpoint, masing-masingnya didefinisikan dan diakses menggunakan empat komponen dalam tabel di bawah. Biasanya, endpoint-endpoint ini dilayani dalam HTML, tetapi karena klien dapat me-request format respon yang lain, maka endpoint-endpoint ini juga bertindak seperti layaknya programmatic API.

2.5.5 RESTful Routing

Dengan adanya routing, aktivitas CRUD pada resource dapat distandarkan dan dicocokkan dengan protokol HTTP dimana pada HTTP juga terdapat aktivitas yang serupa dengan CRUD. Terdapat metode-metode standar HTTP, yang dalam hal ini adalah GET, POST, PUT, DELETE yang akan dipetakan dengan action-action standar dalam controller. Untuk memungkinkan routing seperti ini, maka perlu ditambahkan sebaris kode map.resource :nama_resources pada file konfigurasi routes.rb. Dengan adanya pernyataan seperti itu pada routes.rb, maka Router akan menangani request HTTP yang datang untuk dipetakan/diarahkan pada resource yang sesuai. Ini ditunjukkan pada tabel dibawah ini.

Tabel 2.2 Routing Setiap Metode HTTP dengan Action pada Controller

map.resources :users		
Resource	**Verb**	**Action**
/users	GET	index
/users	POST	create
/users/1	GET	show
/users/1	PUT	update
/users/1	DELETE	destroy



Gambar 2.9 Pemetaan Resource dengan Request pada Router

2.5.6 Kelebihan Web Service

1. Lintas *Platform*

   Penggunaan *web service* memungkinkan komputer-komputer yang berbeda sistem operasi dapat saling bertukar data. Sebagai contoh, jika akan mengakses sebuah *web service*, sistem operasi yang digunakan tidak perlu sama dengan sistem operasi yang digunakan oleh *web service* tersebut.

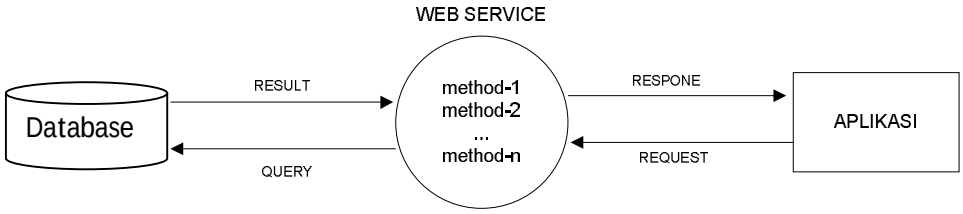
2. *Language Independent*

   Sebuah *web service* dapat diakses menggunakan bahasa pemrograman apa saja. *Web service* yang dibuat dengan PHP bisa diakses oleh berbagai bahasa pemrograman web seperti PHP itu sendiri, JSP hingga bahasa lainnya seperti Delphi, Java, dan VB.NET. *Web service* juga memungkinkan untuk diakses oleh *mobile device* seperti handphone dengan memanfaatkan aplikasi berbasis Java untuk mobile device (*Java 2 Micro Edition* atau lebih dikenal dengan singkatan J2ME)

3. Jembatan Penghubung Dengan *Database*

   Pada umumnya sebuah aplikasi memerlukan *driver database* agar bisa melakukan koneksi ke sebuah database. *Web service* dapat dijadikan sebagai "jembatan" penghubung antara aplikasi dengan database. Jadi dengan

memanfaatkan *web service* sebagai “jembatan ” penghubung, sebuah aplikasi tidak lagi memerlukan *driver database* dan tidak perlu mengetahui database apa yang digunakan oleh *server* serta bagaimana struktur *database* tersebut jika ingin mengaksesnya. Aplikasi tersebut cukup mengetahui *method* atau fungsi apa saja yang disediakan *web service* untuk memanfaatkan fasilitasnya.



Gambar 2.10 *Web Service* Sebagai "Jembatan" Penghubung Aplikasi dengan Database

4. Mempermudah Proses Pertukaran Data

   Penggunaan *web service* dapat mempermudah dan mempercepat pertukaran data diantara mereka daripada mereka harus menyesuaikan aplikasi dan database yang digunakan yang tentu akan memakan waktu dan tenaga yang besar.

5. Penggunaan Kembali Komponen Aplikasi

   Beberapa aplikasi yang berbeda bisa saja memerlukan sebuah fungsi yang sama. Misalnya saja ada aplikasi web dan desktop yang memerlukan sebuah fungsi untuk melakukan konversi kurs antar mata uang yang berbeda.

2.5.6.1 Komponen Pendukung Web Service Extensible Markup Language (XML)

XML termasuk ke dalan keluarga *Markup Language* dan masih satu keluarga dengan HTML (*Hypertext Markup Language*) yang digunakan untuk membuat halaman web. Meskipun masih satu kelurga, kedua bahasa tersebut sebenarnya memilki fungsi yang berbeda. Jika fungsi HTML lebih diutamakan untuk mengatur tampilan, maka fungsi XML lebih ditujukan untuk menyimpan data dan informasi. Selain itu ada satu perbedaan lagi antara HTML dan XML. Di dalam XML kita bisa mendefinisikan tag milik kita sendiri. Contohnya sebagai berikut

```
<anggota>
       <nama> Seseorang</nama>
       <umur>25</umur>
</anggota>
```

Seperti terlihat pada contoh diatas, kita bisa mendefinisikan tag kita sendiri untuk menjelaskan sebuah struktur data bernama anggota yang memiliki atribut nama dan umur.

XML merupakan dasar yang penting atas terbentuknya *Web Service*. *Web service* dapat berkomunikasi dengan aplikasi-aplikasi yang memanggilkanya dengan menggunakn XML. Kenapa menggunakan XML? Karena XML berbentuk teks sehingga mudah untuk ditransportasikan menggunakan protokol HTTP. Selain itu XML juga bersifat *platform independen* sehingga informasi di dalamnya bisa baca oleh aplikasi apapun selama aplikasi tersebut mampu menerjemahkan tag-tag XML

2.6 Pengertian Analisis Kebutuhan Non Fungsional

Mengidentifikasi batasan dari fasilitas yang disediakan oleh sistem. Bisa berupa:

1. Respon time
2. Waktu antara *user* meng*input*kan data dan tampilan yang dihasilkan berdasar *input* data tsb
3. *Security need*
4. Akses *user*

Contoh Akses *user*:

a. Kepala koperasi

   Fungsi pembuatan laporan, fungsi pencarian

b. Bagian keuangan

   Fungsi pembuatan laporan, fungsi pencarian, fungsi pengolahan data simpanan, fungsi pengolahan data pinjaman

c. Bagian kasir

   Fungsi pengolahan data simpanan, fungsi pengolahan data pinjaman, fungsi pencarian

d. Bagian Administrasi

   Fungsi pengolahan data anggota

   (http://www.kuliah.ningyung.com/analisa-kebutuhan/)

2.6.1 Spesifikasi Proses

Spesifikasi Proses menggambarkan kejadian di dalam setiap *bubble* pada level terbawah pada data flow diagram. Spesifikasi proses mendefinisikan

kegiatan yang harus dilakukan untuk mengubah *input* menjadi *output* (*Edward Yourdon, Modern Structured Analysis, hal. 203*).

Sepsifikasi proses digunakan untuk mendeskripsikan proses yang terjadi pada level yang paling dasar dalam DFD. Model ini berfungsi mendeksripsikan apa yang dilakukan ketika masukan ditransformasi menjadi keluaran. Ada berbagai macam *tool*s yang dapat kita gunakan untuk menghasilkan suatu spesifikasi proses: tabel keputusan, Bahasa Inggris terstruktur, pre/post condition, *flowchart*s, diagram Nassishneiderman, dan lain sebagainya. Sedangkan kebanyakan analisis sistem mengarah ke Bahasa Inggris terstruktur, anda harus ingat bahwa setiap metode dapat digunakan, selama hal tersebut memuaskan dua keadaan penting:

1. Spesifikasi proses harus ditampilkan dalam suatu bentuk/form yang dapat diverifikasi oleh *user* dan sistem analis. Kondisi ini tepat untuk alasan ini, dimana kita mengurangi paparan Bahasa Inggris sebagai sebuah alat spesifikasi: hal ini dikenal sebagai ambigu, khususnya apabila menggambarkan tindakan (keputusan) alternatif dan tindakan berulang (loops). Secara alami, hal tersebut juga cenderung menyebabkan kebingungan yang sangat ketika mengekspresikan bagian-bagian kondisi Boolean (misalkan kombinasi dari operator Boolean AND, OR dan NOT).
2. Spesifikasi proses harus ditampilkan dalam suatu bentuk/form yang dapat mengkomunikasikan secara efektif berbagagai keterlibatan berbagai latar belakang pendengar. Sedangkan hal tersebut akan menjadi tipe dari analis sistem yang menuliskan spesifikasi proses, hal tersaebut biasanya menjadi

bermacam-mcam pendengar dari para pengguna, manager, auditor, personil quality assurance, dan lainnya yang membaca spesifikasi proses. Suatu proses spesifikasi diharapkan dapat ditampilkan dalam perhitungan yang dapat diprediksi, atau dalam pascal, atau dalam pendekatan format diagram seperti software use-it; tetapi jika komunitas *user* menolak untuk melihat pada beberapa spesifikasi, mereka adalah tidak berharga.

(*http://dhamidin.files.wordpress.com/2008/01/handout-8.pdf)*

2.6.2 Skema Relasi

Skema relasi (*relation schema*) adalah relasi yang memiliki nama, didefinisikan oleh himpunan pasangan attribute dan domainnya.

Notasi:

R(*A*1, *A*2, … *An*)

R menyatakan nama relasi,

*A*1, *A*2, … *An* menyatakan nama attribute

Contoh:

EMPLOYEE (SSN, FName, LName, Sex, Salary) menyatakan bahwa relasi bernama EMPLOYEE didefinisikan pada 5 attribute: SSN, FName, LName, Sex, Salary)

Domain(SSN) = Number(9), Domain(FName) = Char(10),

Domain(LName) = Char(10), Domain(Sex) = Char(1),

Domain(Salary) = Currency

(http://ocw.ui.ac.id/materials/12.01_FASILKOM/IKI20430-Basis_Data_1/modul_6_model_relasional-Reg.pdf)

2.6.3 Pengujian

2.6.3.1 Definisi

Pengujian adalah proses untuk menemukan *error* pada perangkat lunak sebelum di-*delivery* kepada pengguna.

2.6.3.2 Tujuan Pengujian

1. Pengujian adalah proses menjalankan program dengan maksud mencari kesalahan (*error*)
2. Kasus uji yang baik adalah kasus yang memiliki peluang untuk mendapatkan kesalahan yang belum diketahui
3. Pengujian dikatakan berhasil bila dapat memunculkan kesalahan yang belum diketahui
4. Jadi pengujian yang baik bukan untuk memastikan tidak ada kesalahan tetapi untuk mencari sebanyak mungkin kesalahan yang ada pada program

Pengujian perangkat lunak seharusnya menghabiskan waktu 30% – 40% dari total biaya pembangunan perangkat lunak. Pengujian merupakan bagian dari salah satu *tugas software verification dan validation*, yang merupakan bagian dari *software quality assurance*

Pengujian perangkat lunak mencakup :

a. **Strategi** : Mengintegrasikan metode perancangan kasus uji dalam sekumpulan langkah yang direncanakan
b. **Metode pengujian**, mencakup Perancangan kasus uji dengan menggunakan metode White Box atau Black Box

(http://images.lie2phoet.multiply.multiplycontent.com/attachment/0/S1au3gooCkUAAEDAb3w1/bab_5_pengujian_perangkat_lunak.pdf?key=lie2phoet:journal:45&nmid=311680078)

2.6.4 Diagram Konteks

Diagram konteks adalah sebuah diagram sederhana yang menggambarkan hubungan antara *entity* luar, masukan dan keluaran dari sistem. Diagram konteks direpresentasikan dengan lingkaran tunggal yang mewakili keseluruhan sistem.

2.6.5 DFD

Data Flow Diagram (DFD) adalah suatu model logika data atau proses yang dibuat untuk menggambarkan darimana asal data dan kemana tujuan data yang keluar dari sistem, dimana data disimpan, proses apa yang menghasilkan data tersebut dan interaksi antara data yang tersimpan dan proses yang dikenakan pada data tersebut.

DFD menggambarkan penyimpanan data dan proses yang mentransformasikan data. DFD menunjukan hubungan antara data pada sistem dan proses pada sistem.

DFD merupakan salah satu alat pembuatan model yang sering digunakan, khususnya bila fungsi-fungsi sistem merupakan bagian yang lebih penting dan kompleks dari pada data yang dimanipulasi oleh sistem. Dengan kata lain, DFD adalah alat pembuatan model yang memberikan penekanan hanya pada fungsi sistem.

2.6.6 Diagram E-R

Menurut Budi Sutedjo, ERD adalah peralatan yang berfungsi untuk menggambarkan relasi atau hubungan dari dua *file* atau dua tabel yang dapat digolongkan dalam tiga macam bentuk relasi, yaitu satu – satu (1 – 1), satu – banyak (1 – N), banyak – banyak (N – N). Model *entity relationship* yang berisi komponen-komponen himpunan entitas dan himpunan relasi yang masing-masing dilengkapi dengan atribut–atribut yang merepresentasikan seluruh fakta yang kita tinjau digambarkan lebih sistematis dengan menggunakan Diagram E–R. Pembuatan Diagram E–R meliputi tahap-tahap berikut:

a. Mengidentifikasi dan menetapkan seluruh himpunan entitas yang akan terlihat.

b. Menentukan atribut-atribut *key* dari masing-masing himpunan entitas.

c. Mengidentifikasi dan menetapkan seluruh himpunan relasi di antara himpunan entitas - himpunan entitas yang ada beserta *foreign key* yang terdapat pada relasi tersebut.

d. Menentukan derajat atau kardinalitas relasi untuk setiap himpunan relasi.

e. Melengkapi himpunan entitas dan himpunan relasi dengan atribut-atribut deskriptif (*non key*).

2.6.7 Kamus Data

Kamus data adalah kumpulan elemen-elemen atau symbol-simbol yang digunakan untuk membantu dalam penggambaran atau pengidentifikasian setiap *field* atau *file* di dalam sistem.

Dalam membuat kamus data, harap diperhatikan beberapa hal yaitu :

1. Apakah semua aliran data dan penyimpanan dalam DFD sudah didefinisikan dalam kamus data?
2. Apakah semua komponen elemen data sudah didefinisikan dengan baik?
3. Adakah elemen data yang didefinisikan lebih dari satu kali?
4. Apakah semua notasi yang digunakan pada kamus data sudah dikoreksi?
5. Adakah elemen data dalam kamus data tidak menjelaskan sesuatu dalam DFD atau *Entity Relation*?

2.7 Software Penunjang

2.7.1 MySQL

2.7.1.1 Pengertian MySQL

MySQL adalah perangkat lunak *database server* atau sebut saja *Database Smart*. *Database* ini semakin lama semakin populer. Dengan menggunakan *database* ini, data semakin aman dan berdaya guna. *Database* ini juga banyak dipakai pada web *database* sehingga data semakin terintegrasi antara *database* dekstop dengan *database* web. Untuk menggunakan *database MySQL* harus menginstalasinya dahulu ke komputer.

2.7.1.2 Keistimewaan MySQL

Sebagai *server database* dengan konsep *database* modern, *MySQL* memiliki keistimewaan. Beberapa keistimewaan dimiliki *MySQL* sebagai berikut :

1. *Portability*

   Database MySQL berfungsi dengan stabil tanpa kendala, berarti berlaku pada berbagai sistem operasi seperti Windows, Linux, FreeBSD, Mac OS X *Server*, Solaris, Amiga, HP-Unix, dan lain-lain

2. *Open Source*

   MySQL merupakan *database* open source (gratis), di bawah lisensi GPL sehingga dapat memperoleh dan menggunakannya secara cuma-cuma tanpa membayar sepersen pun.

3. *Multiuser*

   MySQL merupakan *database* yang dapat digunakan untuk menangani beberapa user dalam waktu bersamaan tanpa mengalami masalah. Dan memungkinkan sebuah *database server MySQL* dapat diakses client secara bersamaan pula.

4. *Performance Tuning*

   MySQL mempunyai kecepatan yang cukup baik dalam menangani query-query sederhana, serta mampu memproses lebih banyak *SQL* per satuan waktu.

5. *Column Type*

   Database MySQL didukung dengan tipe data yang sangat kompleks, seperti signed/unsigned *integer*, float, double, char, varchar, text, blob, data, time, datetime, timestamp, year, set serta enum

6. *Command And Functions*

   MySQL server memiliki operator dan fungsi secara penuh yang mendukung perintah SELECT dan WHERE dalam query.

7. *Security*

   Sistem *Security* pada *MySQL* mempunyai beberapa lapisan sekuritas seperti tingkatan *subnetmask, hostname,* dan izin akses user dengan sistem perizinan yang mendetil serta *password* terenkripsi.

8. *Scalability dan Limits*

   MySQL mempunyai kemampuan menangani *database* dalam skala cukup besar, dengan jumlah record lebih dari 50 juta dan 60 ribu tabel serta 5 miliar baris. Selain itu dapat menampung indeks sampai 32 indeks pada tiap tabelnya.

9. *Connectivity*

   Adanya kemampuan *MySQL* melakukan koneksi dengan *client* menggunakan protokol TCP/IP, Unix socket (Unix), atau Named Pipes (NT).

10. *Localization*

    Adanya kemampuan dalam mendeteksi kesalahan (*error code*) pada *client* menggunakan lebih dari dua puluh bahasa.

11. *Interface*

    MySQL memiliki *interface* terhadap berbagai aplikasi dan bahasa pemograman menggunakan fungsi API (*Application Programming Interface*).

12. *Clients dan Tools*

    Database MySQL dilengkapi berbagai tools yang dapat digunakan untuk administrasi *database*.

13. Struktur Tabel

MySQL memiliki struktur tabel yang lebih fleksibel dalam menangani ALTER TABLE dibandingkan *database* lainnya.

2.7.2 PHP MyAdmin

Pengelolaan *database* dengan *MYSQL* harus dilakukan dengan mengetikkan baris-baris perintah yang sesuai (*command line*) untuk setiap maksud tertentu. Jika ingin membuat *database*, pengguna harus mengetikkan baris perintah yang sesuai untuk membuat *database*. Tetapi hal tersebut tentu cukup menyulitkan karena pengguna harus hafal dan mengetikkan perintahnya satu persatu.

Banyak sekali perangkat lunak yang dapat dimanfaatkan untuk mengelola *database* dalam *MySQL*, salah satunya adalah PHP*MyAdmin*. Dengan PHP*MyAdmin* pengguna dapat membuat tabel, mengisi data dan lain-lain dengan mudah tanpa harus hafal perintahnya. Untuk mengaktifkan PHP*MyAdmin* langkah-langkahnya adalah : yang pertama setelah meng-*install XAMPP*, pengguna harus mengaktifkan web *server APACHE* dan *MySQL* dari control panel *XAMPP*. Yang kedua, jalankan di *browser* yang ada *(IE, Mozilla Firefox* atau *Opera)* lalu ketikkan alamat web berikut : http://localhost/PHPmyadmin/ pada address bar lalu tekan Enter. Langkah ketiga apabila telah nampak *interface* (tampilan antar muka) PHP*MyAdmin* pengguna bisa memulainya dengan mengetikkan nama *database*, nama tabel dan seterusnya.