DEPLOYMENT APLIKASI UNTUK *MULTI SERVER* DENGAN MENGGUNAKAN *CAPISTRANO*

Wisnu Uriawan, Adam Faroqi, Hayati
Jurusan Teknik Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Sunan Gunung Djati Bandung Jl. A.H Nasution No. 105 Bandung 40164
[1]juragan.wisnu@gmail.com, [2] adamfaroqi@yahoo.com, hayhay2301@gmail.com

Abstrak

Sebuah aplikasi *web* yang berjalan di beberapa *server* membutuhkan mekanisme *deployment* yang berbeda dengan aplikasi yang hanya berjalan pada satu *server*. Aplikasi dituntut agar dapat disebarkan ke beberapa *server* dalam waktu yang bersamaan. Salah satu *tools* yang dapat menyelesaikan masalah dalam *deployment* aplikasi ke *multi server* yaitu *capistrano.*
Capistrano menawarkan kemampuan melakukan *deployment* aplikasi ke beberapa *server*. Hal ini menarik untuk diteliti bagaimana algoritma yang digunakan, penanganan kegagalan *deployment* dan penanganan *downtime.* Penelitian dilakukan dengan menganalisis proses kerja, dokumentasi dan *source code*-nya. Hasil penelitian menunjukkan *capistrano* menggunakan *multi-threading* dalam menyebarkan aplikasi ke beberapa *server*. Untuk penanganan kegagalan *capistrano* menyediakan fungsi *rollback* agar kegagalan dapat dipelajari dan diperbaiki. Pada saat aplikasi membutuhkan *downtime*, *capistrano* menyediakan fungsi untuk menampilkan halaman *downtime.*

Kata kunci : *deployment, capistrano, multi-threading, rollback, downtime*

1. Pendahuluan

Dewasa ini internet menjadi suatu kebutuhan yang sangat penting bagi seluruh lapisan masyarakat di dunia, baik itu bagi kalangan pelajar, ilmuwan, dan usahawan. Hal ini menyebabkan semakin meningkatnya permintaan akan kebutuhan informasi dalam internet, sehingga trafik dalam internet semakin padat oleh permintaan akan informasi. Semakin meningkatnya trafik dalam internet menyebabkan beban kerja pada *server* penyedia layanan internet tersebut juga meningkat seiring dengan bertambahnya

permintaan yang masuk, sehingga server tersebut akan kelebihan beban dalam waktu yang pendek, terutama untuk server yang menyediakan layanan yang populer. Oleh karena itu, diperlukan *hardware* dan *software* yang mendukung layanan *highly scalable* dan *highly available service*.

Highly scalable dan *highly available service* dapat dijabarkan sebagai berikut:

- *Scalability*, saat *request* yang diarahkan ke *server* meningkat sehingga beban kerja *server* bertambah, kapasitas *server* dapat ditingkatkan dengan cepat dan mudah untuk memenuhi kebutuhan
- *Availability,* layanan secara keseluruhan harus tersedia setiap waktu.
- *Manageability*, walaupun seluruh sistem mungkin secara fisik besar, harus mudah untuk dikelola.
- *Cost-effectiveness,* seluruh bagian sistem harus hemat biaya dalam pembangunan dan pemeliharaan.[1]

Sebuah sistem dengan satu *server* biasanya tidak cukup untuk menangani beban yang meningkat secara cepat. Salah satu penanggulangan beban yang meningkat bisa dilakukan dengan melakukan *upgrade server,* proses *upgrade server* merupakan proses yang kompleks dan rentan terjadi kegagalan aplikasi *(application failure)* ketika proses *upgrade* sedang berjalan. Semakin tinggi kapasitas *server* yang di-*upgrade*, semakin tinggi biaya yang harus dikeluarkan.

Sekelompok *server* yang dihubungkan oleh jaringan yang cepat, muncul akibat arsitektur untuk membangun layanan yang *scalable* dan *highly available,* arsitektur ini dikenal dengan *cluster of servers*. Dalam arsitektur ini beban kerja dari banyak *request* dibagi ke seluruh *server* oleh sebuah *load balancer*. Sehingga jika salah satu server tidak tersedia karena *down, maintenance,* dan lain-lain, maka *request* masih bisa ditangani oleh server lain.

Jika suatu web aplikasi berjalan di beberapa *server*, maka semua aplikasi di semua *server* dituntut untuk menyediakan *resources* yang sama persis seperti kode, *database*, *file* yang di-*upload* oleh *user*.

Penambahan fitur pada aplikasi menuntut hanya sekali *deployment* ke semua *server*, sehingga dibutuhkan suatu mekanisme khusus untuk melaksanakannya. Salah satu solusi yang diteliti yaitu metodologi untuk *deployment* sebuah *web* aplikasi ke semua *server* menggunakan *Capistrano*. *Capistrano* menawakan kemampuan *deployment* aplikasi ke *multi server* dalam satu waktu.

Internet berasal dari kata *Interconnection Networking* yang mempunyai arti hubungan komputer dengan berbagai tipe yang membentuk sistem jaringan yang mencakup seluruh dunia (jaringan komputer global) dengan melalui jalur telekomunikasi seperti telepon, radio link, satelit dan lainnya. *World Wide Web* adalah suatu ruang informasi di mana sumber-sumber daya yang berguna diidentifikasi oleh pengenal global yang disebut *Uniform Resource Indetifier* (URI). *World Wide Web* sering dianggap sama dengan *internet* secara keseluruhan, walaupun sebenarnya ia hanyalah bagian daripadanya.[3]

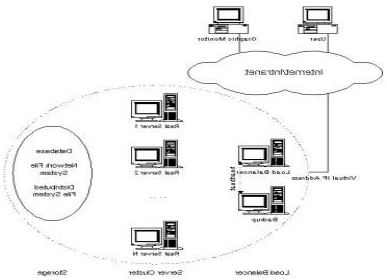
2. Metoda Penelitian

Load balancing adalah suatu metode untuk mendistribusikan beban kepada beberapa *host* sehingga beban kerja menjadi lebih ringan. Ini bertujuan agar waktu rata-rata mengerjakan tugas menjadi singkat dan dapat menaikkan utilitas prosesor. *Load balancing* dapat diimplementasikan dengan *hardware* khusus, software maupun gabungan keduanya. Konfigurasi standar yang ada memberi gambaran bahwa satu mesin ditempatkan diantara *client* dan *server*, mesin ini disebut sebagai *director* karena tugasnya adalah memberikan balancing pada *request* dari *client* ke *server*.

Sebuah load balancer adalah perangkat jaringan yang dipasang diantara *client* dan *server*, bekerja sebagai saklar untuk request dari client. *Load balancer* mengimplementasikan beberapa metode penjadwalan yang akan menentukan ke arah *server* mana *request* dari *client* akan diteruskan.[4]

Deployment means moving your application from a development environment into a home that your customers can visit. For a web application, that process involves choosing a host, setting up a web server and database, and moving all of your files to the right places with the right permissions.[6]

Aplikasi berjalan pada beberapa server yang tergabung menjadi satu virtual server, berikut adalah arsitektur aplikasi multi server meggunakan linux virtual server:



Gambar 1. Arsitektur dasar *multi server*

Arsitektur di atas mengadopsi 3-tier arsitektur yang terdiri atas:

a. *Load Balancer*

Merupakan satu satunya pintu masuk ke dalam sistem/aplikasi. *Load balancer* akan mendistribusikan *request* dari *client* ke server tertentu yang sedang tersedia berdasarkan algoritma tertentu. DNS dari suatu aplikasi web harus diarahkan ke IP dari *load balancer*.

b. *Server Cluster* (kumpulan server)

Kumpulan dari *real server* yang menjadi tempat aplikasi berjalan. Setiap server akan memiliki *resources* yang sama untuk setiap aplikasi. Real server ini lah yang menjadi target deployment jika terdapat penambahan/perbaikan kode aplikasi.

c. *Shared Storage*

Merupakan tempat penyimpanan untuk server-server yang digunakan, seperti file-file yang diupload pengguna dan *database* sehingga seluruh server akan mampu menyediakan konten yang sama.[5]

Version control (*Source Code Management*) adalah sebuah sistem yang mencatat setiap perubahan terhadap sebuah berkas atau kumpulan berkas sehingga pada suatu saat dapat kembali kepada salah satu

versi dari berkas tersebut. *Version control* dapat menyediakan akses untuk banyak orang sekaligus, mirip dengan *sharing folder* yang biasa digunakan, akan tetapi, *version control* memiliki kemampuan yang lebih dari pada sekedar *sharing folder*.[5]

Capistrano is an open source tool for running scripts on multiple servers; its main use is deploying web applications. It automates the process of making a new version of an application available on one or more web servers, including supporting tasks such as changing databases.[8]

Ruby adalah bahasa pemrograman dinamis berbasis skrip yang berorientasi objek. Tujuan dari *ruby* yaitu menggabungkan kelebihan dari semua bahasa-bahasa pemrograman skrip yang ada di dunia. *Ruby* ditulis dengan bahasa pemrograman C dengan kemampuan dasar seperti *Perl* dan *Python*.

Ruby pertama kali dibuat oleh seorang programmer Jepang bernama Yukihiro Matsumoto. Pada tahun 1993 Yukihiro ingin membuat sebuah bahasa skripting yang memiliki kemampuan orientasi objek. Pada saat itu pemrograman berorientasi objek sedang berkembang tetapi belum ada bahasa pemrograman *scripting* yang mendukung pemrograman objek.[6]

3. Pembahasan

Capistrano merupakan sebuah program *ruby open source*, siapa saja boleh berkontribusi dan boleh memakai program ini secara bebas (*free*). Karena seluruh *capistrano* dibangun menggunakan bahasa pemrograman *ruby*, maka sebelum melakukan instalasi di dalam komputer, pengguna harus sudah memasang *ruby* dan *rubygems*. *Rubygems* merupakan sebuah *package manager* yang menyediakan format standar dalam pendistribusian sebuah program dan *library* yang dibangun menggunakan bahasa pemrograman *ruby*. Berikut cara instalasi capistrano melalui *command prompt.*

```
# gem install capistrano
```

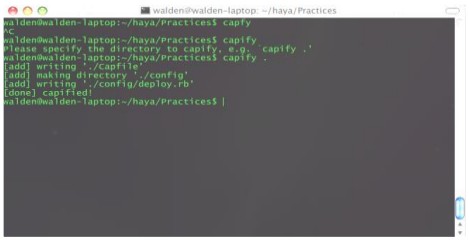
 ISSN 1979-8911

Selain memiliki ketergantungan terhadap instalasi *ruby* dan *rubygems*, dalam penggunannya *capistrano* juga bergantung pada *library ruby* lainnya, yaitu *highline, net-scp, net-sftp, net-ssh dan net-ssh-gateway*. Berbeda dengan *ruby* dan *rubygems* yang harus diinstal terlebih dahulu sebelum instalasi *capistrano, library* tersebut akan secara otomatis terinstal pada saat *capistrano* diinstal.

Capistrano tidak memiliki *GUI* yang dapat digunakan oleh *user*, *user* harus menggunakannya melalui *command prompt.* Terdapat dua jenis perintah dasar yang digunakan dalam *capistrano,* yaitu:

a. "capify"

Jika *capistrano* sudah terinstal, maka ketika diketikkan "capify" pada layar, akan menghasilan dua buah file. File *Capfile* dan file config/deploy.rb. *Capfile* inilah yang akan dibaca oleh capistrano pada saat deployment, lalu *capifile* akan mencari file deploy.rb yang berisi setting dari skenario *deployment.* Berikut ini contoh penggunaan "capify" :



Gambar 2. Contoh penggunaan *capify*

b. "cap"

Perintah "cap" dijalankan pada saat deployment, dan diikuti oleh *task*, contoh "cap deploy". Ada beberapa jenis perintah dasar (*default*) cap yang disediakan oleh *capistrano*, untuk mengetahuinya digunakan script "cap -T", maka akan muncul daftar jenis perintah cap sebagai berikut:

```
cap deploy              # Deploys your project.
cap deploy:check        # Test deployment dependencies.
cap deploy:cleanup      # Clean up old releases.
cap deploy:cold         # Deploys and starts a `cold' application.
```

ISSN 1979-8911

```
cap deploy:create_symlink # Updates the symlink to the most recently deployed...

cap deploy:migrate        # Run the migrate rake task.

cap deploy:migrations     # Deploy and run pending migrations.

cap deploy:pending        # Displays the commits since your last deploy.

cap deploy:pending:diff   # Displays the `diff' since your last deploy.

cap deploy:restart        # Blank task exists as a hook into which to install...

cap deploy:rollback       # Rolls back to a previous version and restarts.

cap deploy:rollback:code  # Rolls back to the previously deployed version.

cap deploy:setup          # Prepares one or more servers for deployment.

cap deploy:start          # Blank task exists as a hook into which to install...

cap deploy:stop           # Blank task exists as a hook into which to install...

cap deploy:symlink        # Deprecated API.

cap deploy:update         # Copies your project and updates the symlink.

cap deploy:update_code    # Copies your project to the remote servers.

cap deploy:upload         # Copy files to the currently deployed version.

cap deploy:web:disable    # Present a maintenance page to visitors.

cap deploy:web:enable     # Makes the application web-accessible again.

cap invoke                # Invoke a single command on the remote servers.

cap shell                 # Begin an interactive Capistrano session.
```

Seperti disebutkan di atas, pada saat deployment aplikasi, capistrano membaca sebuah file konfigurasi, yaitu “Capfile” dan

"deploy.rb". Berikut ini adalah hal-hal yang harus dituliskan pada file "deploy.rb".

a. Nama Aplikasi

```
set :application,
"tutorial"
```

b. Sumber kode aplikasi

```
set :repository,
"repository_url"

set :scm, :subversion

# Atau: `accurev`,
`bzr`, `cvs`, `darcs`,
`git`, `mercurial`,
`perforce`,
`subversion` or `none`

set :scm_username,
"user_name"

set :scm_password,
"password"
```

c. Folder target untuk kode aplikasi

```
set :deploy_to, "folder
target yang ada di
server"
```

d. Server target

```
role :web, "your web-server here"

role :app, "your app-server here"

role :db,  "your primary db-server
here", :primary => true

role :db,  "your slave db-server
here"
```

e. Hal-hal lain yang dibutuhkan pada saat *deployment*

Pada saat sebuah web aplikasi di-*deploy*, biasanya selain memperbaharui kode aplikasi, dibutuhkan langkah-langkah lain sesuai kebutuhan aplikasi, misalnya untuk aplikasi *ruby on rails*, dibutuhkan script untuk me-*restarrt mod_rails* . Dalam hal ini, pengguna hanya perlu menambahkan script pada capistrano. *Script* ini biasanya disebut

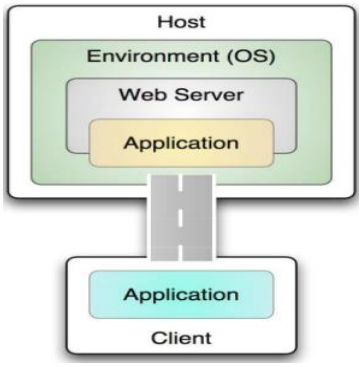
"*task script*", berikut adalah contohnya :

```
namespace :deploy do
  task :start do ; end
  task :stop do ; end
  task :restart, :roles => :app, :except
=> { :no_release => true } do
    run "#{try_sudo} touch

"{File.join(current_path,'tmp','restart.tx
t')}"
  end
end
```

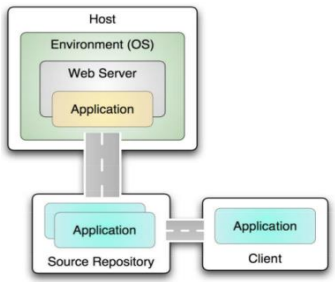
a. Langkah-langkah *deployment*

Deployment diharapkan mampu membuat aplikasi yang telah dibangun di komputer lokal milik pengembang dapat berjalan di server dengan baik. Prosesnya paling sederhana yaitu memindahkan kode sumber ke server, kemudian menjalankannya di server. Sehingga garis besar deployment dapat dipetakan sebagai berikut (*client* merupakan komputer lokal milik pengembang aplikasi):



Gambar 3. *Basic deployment map*

Pada gambar di atas, aplikasi dari komputer pengembang disalin ke server menggunakan mekanisme *copy-paste* yang umum seperti *ftp, scp* atau diunggah menggunakan UI seperti yang disediakan perusahaan penyedia hosting. Pada perkembangannya, banyak pengembang yang menggunakan *source code management* (SCM) dalam pengelolaan kode aplikasi, sehingga *deployment* dipetakan lagi seperti gambar di bawah ini.



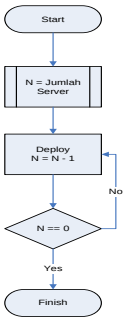
Gambar 4. *Basic deployment map* menggunakan SCM

Sebuah aplikasi yang kodenya dikelola menggunakan SCM, aplikasi tidak disalin ke server dari komputer lokal, melainkan disimpan terlebih dahulu ke *repository* SCM untuk kemudian diunduh secara langsung dari server. Penggunaan SCM akan sangat membantu jika aplikasi dibangun oleh sekelompok orang dalam tim.

b. Iterasi pada *deployment* aplikasi ke banyak server

Pada *deployment* sebuah aplikasi web, jika terdapat sebuah *server* maka terdapat sekali proses *deployment*, jika terdapat dua *server*, maka akan dilakukan dua kali *deployment,* begitu seterusnya sehingga akan didapatkan N iterasi *deployment,* dimana N adalah jumlah *server.*

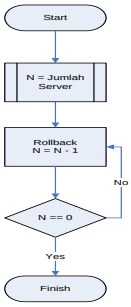
Selanjutnya, iterasi dalam proses deployment dapat digambarkan dengan *flowmap* berikut ini:



Gambar 5. Flowmap proses deployment untuk aplikasi multiserver.

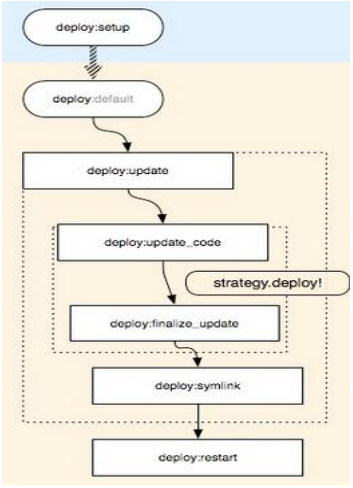
Suatu *deployment* aplikasi, terkadang tidak berhasil atau menimbulkan *error* dan *bugs* yang lebih besar. Dalam hal ini, jika *bugs* tidak dapat ditangani dengan cepat, biasanya langkah terbaik adalah mengembalikan aplikasi ke keadaan sebelum *deployment,* dengan kata lain *rollback deployment.* Proses *rollback* pun akan memiliki N iterasi jika terdapat lebih dari satu server aplikasi.

Proses iterasi penanganan error atau kesalahan *deployment* selanjutnya dapat digambarkan dengan *flowmap* yang berikut ini:



Gambar 6. *Flowmap* proses *rollback deployment* untuk aplikasi *multi server***.**

Pada dasarnya, *capistrano* sudah menyediakan *default deployment behaviour.* Jika proses *deployment* web aplikasi sama seperti yang disediakan oleh *capistrano,* maka tidak diperlukan mendefinisikan *task* baru. Berikut ini adalah *capistrano default deployment behaviour.*



Gambar 7. *Capistrano default deployment behaviour*

4. Kesimpulan

Dari hasil analisis dan pengujian tentang *capistrano*, maka dapat disimpulkan sebagai berikut:

1. *Capistrano* menggunakan *multi-threading* pada proses koneksi ke *server*, sedangkan eksekusi setiap poses pada masing-masing *server* dilakukan secara berurutan satu per satu dengan selisih waktu dalam hitungan *milisecond*.
2. *Capistrano* menyediakan fungsi *deploy:rollback* untuk mengembalikan *deployment* ke versi sebelumnya dalam menangani kegagalan *deployment*..
3. *Capistrano* menyediakan fungsi *deploy:web:disable* pada saat aplikasi membutuhkan *downtime.*

5. Saran

Dengan penelitian ini, dapat diambil beberapa saran sebagai berikut:

1. Setelah menggunakan fungsi *deploy:rollback* perlu dilakukan peninjauan kembali terhadap penyebab kegagalan dan dibuatkan solusi yang

lebih baik agar kegagalan tidak terjadi pada *deployment* selanjutnya.

2. Pada penggunaan fungsi *deploy:web:disable*, diperlukan konfigurasi tambahan pada *web server*.

DAFTAR PUSTAKA

[1] Chendramata, Aidil & Adhityo P. 2008 . *High Availibity System*, Jakarta: Direktorat Sistem Informasi, Perangkat Lunak dan Konten irektorat Jendral Aplikasi Telematika, Departem en Komunikasi dan Informatika.

[2] Dhanta, Rizky. 2009. *Kamus Istilah Komputer Grafis dan.* Surabaya : Indah

[3] Khoe Yao Tung.1997.*Teknologi Jaringan Internet.*Yogyakarta : Dinastindo

[4] Y.M., dan Rassul Ayani. 2001. *Comparison of Load balancing Strategies on Cluster-based Web Servers*. Singapore : Fujitsu Computer (Pte) Ltd & National University of Singapore.

[5] Zhang, W. *Linux Virtual Server for Scalable Network Services*. China : National Laboratory for Parallel & Distributed Processing.

[6] Zygmuntowicz, Ezra dkk.2007. *Deploying Rails Application Step by Step.*Texas:The Pragmatic Bookshelf

[7] http://www.linuxvirtualserver.org (di akses tanggal 31 Juli 2012)

[8] http://en.wikipedia.org/wiki/Capistrano (diakses tanggal 31 Juli 2012)

[9] https://github.com/capistrano/capistrano/wiki/_pages(diakses tanggal 31 Juli 2012)

[10] https://github.com/capistrano/capistrano.git(diakses tanggal 31 Juli 2012)