Proyecto de Fin de Carrera

Pizarra Web Compartida

Albert Llop

19 de diciembre de 2008

Índice general

Capítulo 1

Deployment

1.1. Introducción

El desarrollo de aplicaciones web mediante Frameworks del estilo de Ruby on Rails necesita de ciertas medidas a la hora de realizar su `deploy` (instalación y puesta en marcha) en los servidores donde se va a acabar sirviendo la aplicación finalmente. Existen múltiples posibilidades, pero siempre hay que tener en cuenta que, las aplicaciones, para poder funcionar, necesitan cargar una serie de librerías en memoria, que son la que realizan todo el trabajo. En este capítulo se comentarán las posibilidades existenten a la hora de realizar el llamado deploy de una aplicación Rails, y más específicamente ésta.

1.2. Configuración básica del servidor

Ruby es un lenguaje bastante novedoso, y aunque cada vez más las distribuciones modernas de sistemas UNIX tran por defecto soporte para este lenguaje, es posible que en instalaciones clásicas dicho soporte no exista. Por tanto, es necesario realizar una instalación previa, necesaria para cualquier técnica usada a posteriori para servir la aplicación. En todos los pasos se incluirán opciones alternativas para las distribuciones que utilizan APT (como Debian y Ubuntu) y las que utilizan RPM (mediante YUM, como Fedora o CentOS, distribuciones muy comunes en configuraciones de servidores web).

1.2.1. Ruby

Es posible instalar el soporte para Ruby de múltiples maneras [1]. El objetivo, sea cual sea el procedimiento, es que al ajecutar la línea `ruby -v` se obtenga una línea del estilo de:

```
$ ruby -v
ruby 1.8.7 ...
```

Compilando las fuentes

La primera de ellas, compilando las fuentes :

```
$ wget http://ftp.ruby-lang.org/pub/ruby/1.8/ruby-1.8.7.tar.gz
$ tar xvzf ruby-1.8.7.tar.gz
$ cd ruby-1.8.7.tar.gz
$ ./configure
$ make
# make install
```

A partir de aquí, es recomendable realizar un enlace simbólico de forma que el comando `ruby` ejecute esta versión, puesto que suele instalarse como `ruby1.8`

APT

Es necesario instalar los siguientes paquetes.

```
# apt-get install ruby1.8-dev ruby1.8 ri1.8 rdoc1.8 irb1.8 \
                  libreadline-ruby1.8 libruby1.8 libopenssl-ruby
```

[1] A fecha de escritura de este capítulo, Ruby on Rails recomienda el uso de Ruby 1.8.7, aunque las versiones 1.8.6, 1.8.5 y 1.8.4 son funcionales. http://rubyonrails.org/down

RPM

Existe un repositorio creado por la comunidad, con diferentes paquetes necesarios para el soporte de ruby en sistemas compatibles con YUM. Para añadir este repositorio, editar el archivo `/etc/yum.repos.d/ruby.repo` y añadir:

```
[ruby]
name=ruby
baseurl=http://repo.premiumhelp.eu/ruby/
gpgcheck=0
enabled=1
```

Posteriormente es posible instalar los paquetes necesarios:

```
# yum install ruby ruby-devel ruby-docs
```

En caso de no estar disponible dicho repositorio, sería fácil encontrar dichos RPM's en algún otro repositorio. Los binarios incluidos en este repositorio a fecha de la escritura de este documento, referentes al soporte de Ruby son los siguientes:

```
ruby-1.8.6.111-1.i686.rpm
ruby-devel-1.8.6.111-1.i686.rpm
ruby-docs-1.8.6.111-1.i686.rpm
ruby-irb-1.8.6.111-1.i686.rpm
ruby-libs-1.8.6.111-1.i686.rpm
ruby-mode-1.8.6.111-1.i686.rpm
ruby-mysql-2.7.4-1.i686.rpm
ruby-postgres-0.7.1-6.i686.rpm
ruby-rdoc-1.8.6.111-1.i686.rpm
ruby-ri-1.8.6.111-1.i686.rpm
ruby-tcltk-1.8.6.111-1.i686.rpm
```

1.2.2. RubyGems

Una Gem es el formato standard para distribuir librerías de todo tipo para Ruby. RubyGems es la vía standard de distribución de dichas Gems, y funciona de una forma similar a APT o YUM, en cuanto a que es posible buscar, instalar y desinstalar Gems mediante instrucciones en la línea de comandos, descargando automáticamente los archivos necesarios de los repositorios públicos.

En este caso, y teniendo instalado el soporte para Ruby, para instalar RubyGems es necesario bajar la última versión de la web oficial del proyecto [2], y instalar mediante [3]:

```
$ tar xvzf rubygems-1.3.1.tgz
$ cd rubygems-1.3.1.tgz
# ruby setup.rb --no-rdoc --no-ri
```

RubyGems se instalará automáticamente en el sistema, pudiéndose comprobar con una instrucción semejante a la necesaria con Ruby:

[2] http://rubyforge.org/projects/rubygems/

[3] A fecha de la escritura de este capítulo, la última versión de RubyGems es 1.3.1

```
$ gem -v
1.3.1
```

Una vez instalado RubyGems, suele ser necesario instalar las Gems que use la aplicación que se quiere hacer funcionar. En el caso de esta aplicación, las gemas necesarias son las siguientes:

```
# gem install RedCloth haml mime-types rmagick mongrel mongrel_cluster rack rails rake rs
```

rails Oficialmente, no debería ser imprescindible instalar esta Gem en el servidor, puesto que la práctica habitual es *congelar* la versión de Rails utilizada para desarrollar la aplicación, y así evitar que la versión actual de la aplicación se rompa con versiones futuras de la Gem. Es una política usada por Rails para librerarse de la carga de mantener soporte con versiones antiguas. Realísticamente, instalar Rails es la forma más fácil de instalar todas las dependencias básicas de cada aplicación en Rails.

rake Se instala automáticamente con la rails Gem. Rake, o *Ruby Make*, es el Make que se usa en cualquier aplicación en Ruby. Su sintaxis es en Ruby directamente, y existen múltiples *tareas* ya disponibles para realizar lo más común, como crear la base de datos y toda su estructura a partir de las *migraciones*, rotar Logs, congelar la rails Gem, además de otras tareas personalizadas, como se explicará más adelante.

rspec RSpec es un *Behaviour Driven Development Framework for Ruby* (o BDD framework), utilizado por Rails, y que se instala automáticamente por la rails Gem. Un BDD framework provee un método sencillo de traducir los requisitos impuestos por las funcionalidades, a una serie de Tests que comprueban que el código escrito hace lo que se ha especificado que tiene que hacer. Dichos tests se pueden ejecutar en la instalación de cualquier aplicación Rails para comprobar que ninguna modificación añadida por el código de la aplicación, o de algun Plugin o Gem, no haya *roto* Rails.

RedCloth A la hora de introducir descripciones, tanto para los grupos como para los documentos, se permite un formateo limitado del texto mediante Textile [4], una sintaxis de escritura semejante a Markdown [5]. RedCloth se utiliza para convertir texto con formato Textile a html de forma segura evitando posibles códigos maliciosos.

haml Haml es una alternativa para implementar vistas para Rails. La forma tradicional es mediante ERB, el cual permite incrustar código ruby dentro de plantillas normales en HTML, de forma muy similar a como se incrusta código al programar webs con PHP o JSP. Haml ofrece una sintaxis alternativa a HTML, utilizando la tabulación para controlar el cierre de etiquetas, eliminando una gran cantidad del código y integrándose en mayor forma con Ruby. Algunas de las vistas de la aplicación están implementadas en haml.

mime-types Esta Gem permite obtener el mime-type de prácticamente cualquier archivo con formato conocido. Necesaria a la hora de generar objetos de tipo Image cuando se tiene una imagen en el disco duro del sistema.

rmagick Un interface escrito en Ruby para ImageMagick. Necesario para el plugin `attachment-fu`. `MiniMagick` o `ImageScience` serían alternativas a esta Gem.

mongrel Mongrel es un servidor web muy sencillo escrito en Ruby, y que se puede usar para arrancar instancias de la aplicación.

[4] http://www.textism.com/tools/textile/
[5] http://daringfireball.net/projects/markdown/

mongrel_cluster Es un plugin para simplificar el proceso de arrancar múltiples instancias de servidores Mongrel a la vez.

1.3. De desarrollo a producción

Las necesidades de una aplicación funcionando en local, en un entorno de desarrollo donde lo que se necesita es realizar pruebas, no son las mismas que las de una aplicación funcionando en el servidor final, ante usuarios que no son los desarrolladores.

Así pues, una aplicación rails viene preparada para comportarse de forma distinta en ambas situaciones. Entre otras, las diferencias básicas más importantes son las siguientes:

- En modo desarrollo, cuando se produce un error, se genera todo un informe de error con la información necesaria para solucionarlo. En producción, los usuarios no necesitan ver ninguna línea de código, por tanto siempre se muestra un mensaje de error genérico, del tipo 404 o 500.
- Cacheo de clases. Cuando se está trabajando en local lo más cómodo es poder realizar cambios en el código que se reflejen inmediatamente en el servidor local con que se está trabajando. En producción, sin embargo, no se espera que hayan cambios en el código, manteniendo por tanto todas las clases cargadas en memoria, sin que se vuelvan a cargar los archivos en memoria hasta que no se reinicie el servidor.

El segundo punto hace que las aplicaciones puedan funcionar de forma muy eficiente en situaciones en que la aplicación puede mantenerse en memoria.

1.4. FastCGI

CGI, o *Common Gateway Interface* es un protocolo para permitir la comunicación entre servidores web y aplicaciones funcionando en procesos separados, que se inician al principio de una petición web, y se paran al servir la petición. FastCGI es una variación de CGI que lo mejora en varios aspectos.

Mediante CGI es posible ejecutar cualquier tipo de proceso desde un servidor web, y suele ser la forma de tener funcionando scripts en lenguajes varios, como Perl o Python. También es posible ejecutar código PHP, por ejemplo, pero existen otras alternativas para PHP dada la popularidad del lenguaje en entornos web, más específicas y por tanto, más eficientes.

CGI se ha utilizado desde el principio para servir código en Ruby, y toda aplicación en Ruby on Rails viene iniciada con soporte para CGI y FastCGI. Generalmente, realizar un deploy que pretenda servir la aplicación mediante FastCGI solo necesita que la configuración del site en apache, el atributo `DocumentRoot` apunte a la carpeta `/public` de la aplicación, y no a la raíz. Por ejemplo, una configuración mínima sería:

```
<VirtualHost *:80>
    DocumentRoot /var/www/ejemplo/public
    Options Indexes ExecCGI FollowSymLinks
    RewriteEngine On
</VirtualHost>
```

Esta técnica se considera actualmente desfasada, puesto que la eficiencia y escalabilidad son mínimas. Ante cada peticion se carga el entorno rails con las partes necesarias, se genera

la página, y se manda. No se beneficia en absoluto de las características comentadas de una aplicación funcionando en producción en cuanto a eficiencia se refiere.

Actualmente se utiliza en algunos servidores de hosting compartidos, donde no se permiten acciones más complejas como el arranque de procesos por parte del cliente. Incluso en estos casos, en la actualidad existen opciones más adecuadas para estas situaciones, como se verá más adelante en la sección sobre Phussion Passenger 1.5 .

1.5. Phussion Passenger

Phussion Passenger, también conocido como `mod_rails`, es un módulo para el servidor web Apache que añade soporte para aplicaciones escritas en Ruby on Rails (y otros frameworks en ruby con ligeras modificaciones). `mod_rails` es el substituto natural de FastCGI, mejorando substancialmente la eficiencia y la escalabilidad de las aplicaciones funcionando en este entorno.

A diferencia de FastCGI, `mod_rails` carga en memoria instancias de la aplicación de forma dinámica según la demanda de la aplicación en si, siempre dentro de unos parámetros establecidos. Además, permite mantener cargado en memoria el núcleo de Rails en si, de forma que las diferentes aplicaciones de un mismo servidor que utilicen `mod_rails` para servir sus páginas, compartirán dicho núcleo.

Gracias a esta capacidad de compartir recursos entre aplicaciones, es la solución ideal para máquinas que tendrán varias aplicaciones en Rails funcionando al mismo tiempo, produciendo ganancias en cuanto a memoria, y manteniendo un rendimiento prácticamente equivalente a otras soluciones.

Para instalar `mod_rails` en un servidor con apache, se debe instalar su Gem:

```
# gem install passenger
```

Y ejecutar el instalador:

```
# passenger-install-apache2-module
```

El instalador compila automáticamente las fuentes necesarias e informa de los paquetes faltantes, sugeriendo los comandos a realizar para instalarlos según la distribución de Linux en que se encuentre. Al final, indica tres líneas que deben añadirse al archivo de configuración de apache.

Desde ese momento, hay que realizar los mismos pasos que con FastCGI, haciendo que el site configurado apunte su `DocumentRoot` a la carpeta `/public`

1.6. Mongrels o Thins

Existen otro tipo de enfoques para servir aplicaciones rails que permiten una configuración mucho más personalizada. Tanto `Mongrel` como `Thin` son servidores web muy sencillos y ligeros especializados en servir una aplicación rails. Es posible arrancar cualquiera de los dos desde una aplicación rails, ligada a un puerto, y desde ese momento se puede acceder a esa aplicación desde ese puerto, sin necesidad de más configuración.

Ambos mantienen todo el tiempo la aplicación cargada en memoria, y aprovechan todas las características de una aplicación Rails funcionando en producción. Sin embargo, y como se ha comentado con anterioridad, una instancia de dichos servidores solo puede servir una página a la vez, por lo tanto se recomienda tener múltiples instancias dependiendo de la carga de dicha aplicación.

Ambos pueden instalarse en forma de Gem, y lo aconsejable es crear un archivo de configuración en el que guardar la configuración con la que se desean arrancar las diferentes instancias.

```
# mongrel_config.yml
pid_file: tmp/pids/mongrel.pid
log_file: log/mongrel.log
port: "3000"
environment: production
cwd: /path/to/app
address: 0.0.0.0
servers: 2

# thin_config.yml
pid: tmp/pids/thin.pid
log: log/thin.log
port: "3000"
environment: production
chdir: /path/to/app
address: 0.0.0.0
servers: 2
daemonize: true
```

En ambos casos se pretenden levantar dos instancias, empezando en el puerto 3000, por lo tanto ocuparán los puertos 3000 y 3001. Se pueden arrancar dichas instancias con los siguientes comandos.

```
$ mongrel_rails cluster::start -C path/to/mongrel_config.yml
$ thin start --all path/to/thin_config.yml
```

Una vez se tienen varias instancias funcionando, se necesita un balanceador de carga de modo que se vayan distribuyendo las peticiones entre todas las instancias. El proceso tradicional para esto es, mediante apache, configurar el site de la siguiente manera:

```
<VirtualHost *:80>
    DocumentRoot /path/to/application/public

    RewriteEngine On

    <Proxy balancer://app>
      BalancerMember http://127.0.0.1:3000
      BalancerMember http://127.0.0.1:3001
    </Proxy>

    # Redirect all non-static requests
    RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
    RewriteRule ^/(.*)$ balancer://app%{REQUEST_URI} [P,QSA,L]

    ProxyPass / balancer://app/
    ProxyPassReverse / balancer://app/
    ProxyPreserveHost on

</VirtualHost>
```

Un par de puntos a destacar:

- Se deben especificar todos las instancias de forma manual, haciendo que la modificación de la cantidad de instancias no se aun proceso automático.
- Las dos líneas de `RewriteCond` y `RewriteRule` comprueban si la solicitud coincide con contenido estático, como podrían ser imágenes, y en caso de ser así, trata la solicitud directamente sin llegar a requerir la acción de una instancia de la aplicación. Este proceso incrementa en gran medida la disponibilidad de las instancias de aplicación, puesto que solo requieren de su acción en los casos en los que realmente es necesario. Si no se incluyera, las instancias deberían atender las peticiones de la página en si (código HTML generado), sino tambien todos y cada uno de los contenidos estáticos (imágenes, archivos css, javascript, etc) los cuales no requieren de procesado, ni de consulta de base dedatos, pero si mantienen la instancia ocupada mientras se transmiten.

1.7. Esta aplicación en particular

La aplicación está online de forma pública [6] desde el hosting de repositorios GIT github [7]. Es posible descargar el código mediante el enlace de descarga incluido en la página de la aplicación, o descargando el código directamente mediante git:

```
$ git clone git://github.com/albertllop/pfc.git
```

Se recomienda la segunda metodología para facilitar una actualización futura más sencilla del código. La mayoría de pasos a seguir a partir de aquí son comunes a cualquier otra aplicación en Ruby on Rails.

1.7.1. Configurar la base de datos

Es necesario especificar los datos para la conexión a una base de datos. El archivo en cuestión se encuentra en `/config/database.yml` y se incluye un archivo de ejemplo en `/config/database.yml.templa`

```
# database.yml.template
development:
  adapter: mysql
  encoding: utf8
  database: drawme_development
  username: root
  password: root
  host: localhost
  socket: /temp/mysql.sock

production:
  adapter: mysql
  encoding: utf8
  database: drawme_production
  username: root
  password: root
  host: localhost
  socket: /temp/mysql.sock
```

Como se observa, es posible configurar diferentes conexiones y bases de datos tanto para el entorno de desarrollo como para el de producción. Ruby on Rails soporta MySql, PostgreSQL y SQLite de forma nativa, aunque existen múltiples Gems y tutoriales para poder conectar con la mayoría de SGBD existentes [8], como Oracle, Microsoft SQL Server o IBM DB2.

Una vez configurada la base de datos todos los procesos a seguir se pueden ejecutar desde la línea de comandos. Se debe crear la base de datos en si, y generar la estructura de tablas:

```
rake db:create RAILS_ENV=production
rake db:migrate RAILS_ENV=production
```

El parámetro `RAILS_ENV=production` es necesario cuando se quiere ejecutar cualquier tarea de `rake` suponiendo el entorno de producción, puesto que el entorno por defecto es el de desarrollo.

[6] http://github.com/albertllop/pfc
[7] http://github.com
[8] http://wiki.rubyonrails.com/rails/pages/HowtosDatabase

1.7.2. Arrancando los mongrels

Puesto que la combinación de Balanceador Proxy de Apache con varios procesos Mongrel es una de las formas más habituales de servir aplicaciones Ruby on Rails, se han incluido unas tareas `Rake` para ayudar con el proceso. Se debe editar el archivo de configuración `/config/mongrel_cluster.yml` y cambiar dos valores, `port` y `servers`:

```
# mongrel_cluster.yml
---
port: 3000
servers: 2
log_file: log/mongrel.log
environment: production
pid_file: tmp/pids/mongrel.pid
```

El parámetro `port` sirve para indicar el puerto en que iniciar los procesos, y el parámetro `servers` para indicar cuantos procesos iniciar. En este caso, se iniciarían dos instancias mongrel en los puertos `3000` y `3001`.

Las tareas en si son las siguientes:

```
rake servers:start
rake servers:stop
rake servers:restart
```

Las cuales inician, paran, o reinician los procesos automáticamente.

1.7.3. Configurando SSL

Una aplicación Ruby on Rails no difiere de cualquier otra en cuanto a su soporte para SSL. Dado que, en los ejemplos contemplados, se presupone una configuración con Apache recibiendo las peticiones (tanto con FastCGI, mod_`rails` o Mongrels), es posible implementar el soporte para conexiones seguras directamente en la configuración del site de Apache. El procedimiento más sencillo es duplicando la configuración incluyendo el puerto de SSL y los parámetros adecuados.

```
<VirtualHost *:443>
  SSLEngine on
  SSLCertificateFile /etc/ssl/certs/cert.pem

  DocumentRoot /path/to/application/public

  RewriteEngine On

  <Proxy balancer://app>
    BalancerMember http://127.0.0.1:3000
    BalancerMember http://127.0.0.1:3001
  </Proxy>

  # Redirect all non-static requests
  RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
  RewriteRule ^/(.*)$ balancer://app%{REQUEST_URI} [P,QSA,L]
```

```
    ProxyPass / balancer://app/
    ProxyPassReverse / balancer://app/
    ProxyPreserveHost on

</VirtualHost>
```

De este modo se tendría soporte en toda la aplicación para SSL. Puesto que se ha establecido que el soporte SSL sería opcional, no se requieren más pasos. En caso de querer añadir SSL como un requisito para usar alguna funcionalidad de la web, o que esté disponible solo en algunas acciones, se podría realizar mediante el plugin `ssl_requirement` [9], soportado oficialmente por Ruby on Rails, pero no incluído en el núcleo de funcionalidades de Rails.

```
# Instalar el plugin
$ script/plugin install ssl_requirement

# Requerir SSL para las acciones login y panel del controlador user_controller.rb
ssl_required :login, :panel

# Ofrecer la posibilidad de utilizar SSL para las acciones login y panel del
# controlador user_controller.rb
ssl_allowed :login, :panel
```

[9] http://dev.rubyonrails.org/svn/rails/plugins/ssl_requirement/