Rails HOW-TO: Apache and Basic Authentication

Barry Cornelius, Oxford University Computing Services

Date: 2008-06-05

1 Introduction

This document describes one way of running Rails applications through the Apache httpd web server. It describes the use of an Apache module called `passenger`.

The document also describes how you can get a Rails application to ask for a valid username and password before running the Rails application. For this, HTTP Basic Authentication is used.

The document assumes that a Debian etch platform is being used. This can be provided by following the commands in the document *Rails HOW-TO: Installing Rails into Debian* which is available at `http://www.oucs.ox.ac.uk/rails/howtos`. Having said that, much of what is described here will work with any Linux distribution.

2 Deploying Rails applications

2.1 The problem

Ruby comes with WEBrick, a web server that can be used to run Ruby applications. It's not very fast and it can only handle one request at a time. So if it's already processing a request, then a request from some other browser will have to sit there waiting for the first one to finish.

2.2 An often-used solution

Mongrel is another web server for Ruby applications; although it also can only handle one request, it processes a request a lot faster then WEBrick.

The Wikipedia page about Mongrel says:

'One popular configuration is to run Apache 2.2 as a load balancer using `mod_proxy_balancer` in conjunction with several Mongrel instances, with each Mongrel instance running on a separate port. This is something that can be configured very easily using the mongrel_cluster management utility. Apache can divide the incoming requests among the available Mongrel processes, and, with careful configuration, can even serve static content itself without having to delegate to Mongrel.'

One easy-to-read document about how to do all of this is at: `http://mongrel.rubyforge.org/wiki/Apache`

2.3 Passenger, the new kid on the block

In April 2008, Phusion, a small company in the Netherlands, released an open source product called *Passenger* (also known as `mod_rails`).

The web page at `http://railsforphp.com/2008/05/13/deploying-on-phusion-passenger/` says:

'Passenger aims to take the complexity out of deploying Rails applications by also integrating with Apache.'

'Installing Passenger on the average Linux or Mac server is usually simple. It is installed as a gem and contains an automated installer program that compiles and installs the necessary components. Once installed, many Rails applications can be deployed under Apache simply by configuring a VirtualHost for each application. The installer even outputs an example for you to copy and paste.'

'Passenger gives a deployment experience closer to what we've come to appreciate with PHP. In the background, there's still more moving parts, but Passenger automatically manages them. It spawns Rails application server processes, proxies to them, and largely eliminates the configuration and glue that other Rails deployment options leave up to you.'

2.4 Using Passenger instead of Mongrel

Although Mongrel and Apache with `mod_proxy_balancer` is a solution that is often used, in this document we will use Passenger.

3 Using Apache and Passenger

3.1 Installing and testing Apache

3.1.1 Installing Apache

The first task is to install the Apache httpd web server. Instructions for doing this are given below.

There are a few different Debian packages for the Apache httpd web server: I use `apache2-mpm-prefork`. Passenger needs the Apache development header files (such as `httpd.h`). For this reason, we also need to install `apache2-prefork-dev`.

If, like me, you are installing into a chroot area, then the installation of Apache will fail to start the Apache web server if port 80 is already in use. So below we get it to use different ports (say 8110 and 8113) rather than 80 (and 443). We then restart Apache.

```
# Apache fails to start if port 80 is already in use
apt-get install apache2-mpm-prefork apache2-prefork-dev
# arrange for Apache to start when rebooted
ed /etc/default/apache2 <<'%'
g/NO_START=1/s//NO_START=0/gp
w
q
%
# listen on ports 8110 and 8113 rather than 80 (and 443)
ed /etc/apache2/ports.conf <<'%'
g/80/s//8110/gp
1a
Listen 8113
.
w
q
%
/etc/init.d/apache2 stop # this might fail
/etc/init.d/apache2 start
```

3.1.2 Testing Apache

By default, Apache uses `/var/www` as its document root. So we now create a file in that directory and access it with a web browser. This is a simple check that the Apache web server is working.

```
cat >/var/www/hello.html <<%
<html>
  <body>
    <p>hello</p>
  </body>
</html>
%
http://www.abcd.ox.ac.uk:8110/hello.html
rm /var/www/hello.html
```

3.2 Installing and testing Passenger

3.2.1 Installing Passenger

A useful guide to installing and using Passenger, the *Passenger users guide*, is available at `http://www.modrails.com/documentation/Users%20guide.html`.

It gives two ways of installing Passenger. The easiest way is to install a RubyGem called `passenger`. For me, this produced a lot of scary messages.

```
apt-get install ruby1.8-dev
gem install passenger
```

Having installed the RubyGem, the `passenger-install-apache2-module` binary will do the rest of the installation of Passenger for you. When you run this binary, you will find that it is very helpful: if you don't have some of the required software (such as that in `apache2-prefork-dev`) it will tell you what to do.

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

`passenger-install-apache2-module` also advises you how to alter the Apache configuration file (either `apache2.conf` or `httpd.conf`). I chose instead to provide files called `passenger.load` and `passenger.conf` in the directory `/etc/apache2/mods-available`.

I have included an additional line not mentioned in the output from `passenger-install-apache2-module`. This is the `RailsEnv` line. This will ensure that, when any Rails application runs, it will be run using the development version (rather than the production or test version) of the database. (There is more about this in the *Configuring Passenger* section of the Passenger users guide.)

```
cd /etc/apache2/mods-available
cat >passenger.conf <<%
RailsSpawnServer \
 /usr/lib/ruby/gems/1.8/gems/passenger-1.0.5/bin/passenger-spawn-server
RailsRuby /usr/bin/ruby1.8
RailsEnv development
%
cat >passenger.load <<%
LoadModule passenger_module \
 /usr/lib/ruby/gems/1.8/gems/passenger-1.0.5/ext/apache2/mod_passenger.so
%
```

The following commands ensure that the Passenger module gets included when Apache is next restarted.

```
cd /etc/apache2/mods-enabled
ln -s /etc/apache2/mods-available/passenger.conf .
ln -s /etc/apache2/mods-available/passenger.load .
```

The Passenger users guide has a section entitled *Deploying a Ruby on Rails application*. This describes two ways of arranging for this Apache-Passenger combination to deliver the execution of several Rails applications.

I'm going to introduce a new directory in Apache's document root directory for this work on Rails. It's called `/var/www/apps`. In that directory, I'm going to provide a symbolic link for each Rails application to the application's `public` directory.

```
mkdir -p /var/www/apps
ln -s /var/apps/contacts/public /var/www/apps/contacts
```

We also need to ensure that the Apache configuration file (`/etc/apache2/sites-available/default`) has the following configuration for the directory `/var/www/apps`:

```
  Options FollowSymLinks
```

The default is for Passenger to run in its clever mode where, for each request to the Apache web server, it looks to see whether it is going to have to run a Rail application. However, I will switch this off; so for each Rails application I will need to provide a `RailsBaseURI` line in the configuration file. Again, see the Passenger users guide for more details.

```
cd /etc/apache2/sites-available
ed default <<'%'
$i
    RailsAutoDetect off
    RailsBaseURI /apps/contacts
.
w
y
%
```

3.2.2 Testing Passenger

In order to test this out, we first need to restart the Apache web server.

```
/etc/init.d/apache2 stop
/etc/init.d/apache2 start
```

We can test this by going to the following URL.

```
http://www.abcd.ox.ac.uk:8110/apps/contacts/phones
```

So we can now use Apache to execute our Rails applications.

4 HTTP Basic Authentication

4.1 Enabling access using https

We now look at how we can get a Rails application to ask for a username and password before it allows access to the rest of the Rails application.

We first need to get the Apache web server to handle https connections.

If you want to add SSL support to an Apache web server, then, whenever a browser contacts your server using `https`, the server must offer a certificate. With some versions of Linux, there is a command called `apache2-ssl-certificate` or `make-ssl-cert` that can be used to create a self-signed certificate.

If you do use a self-signed certificate, a visitor will get a warning message when they use `https` to your server. The alternative is to generate a *certificate signing request* and have this signed by a recognised certificate authority, such as GlobalSign (`http://www.ja.net/cert/web/globalsign.html`). Other useful information is available at `http://www.openssl.org/docs/HOWTO/certificates.txt`, `http://www-128.ibm.com/developerworks/java/library/j-certgen/`. At the University of Oxford, IT Support Staff should look at `https://wiki.oucs.ox.ac.uk/itss/CertificateService`.

Below, the command `make-ssl-cert` is used to create a certificate in `/etc/apache2/ssl/apache.pem`.

```
mkdir -p /etc/apache2/ssl
apt-get install ssl-cert
/usr/sbin/make-ssl-cert /usr/share/ssl-cert/ssleay.cnf \
                        /etc/apache2/ssl/apache.pem
GB
Oxfordshire
Oxford
University of Oxford
The ABCD Department
www.abcd.ox.ac.uk
pat.lee@abcd.ox.ac.uk
ls -lrt /etc/apache2/ssl/apache.pem
```

We need to ensure that the SSL library gets included as part of the Apache web server when it is next restarted.

```
cd /etc/apache2/mods-enabled
ln -s /etc/apache2/mods-available/ssl.conf .
ln -s /etc/apache2/mods-available/ssl.load .
```

The following commands can be used to create a virtual host for https connections on port 8113.

```
cd /etc/apache2/sites-available
cp -p default ssl
ed ssl <<'%'
g/^NameVirtualHost \*/s//NameVirtualHost *:8113/p
g/^<VirtualHost \*>/s//<VirtualHost *:8113>/p
$i
    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/apache.pem
.
w
q
%
```

The file `ssl` now looks like the following.

```
NameVirtualHost *:8113
<VirtualHost *:8113>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www
...
    RailsAutoDetect off
    RailsBaseURI /apps/contacts
    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/apache.pem
</VirtualHost>
```

We now add the new virtual host to the server, restart Apache and test it now accepts https to port 8113.

```
cd /etc/apache2/sites-enabled
ln -s /etc/apache2/sites-available/ssl 000-ssl
/etc/init.d/apache2 stop
/etc/init.d/apache2 start
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones
http://www.abcd.ox.ac.uk:8110/apps/contacts/phones
```

4.2 Denying access to http

In order to ensure that all accesses to the Rails application are through https, we will remove access through http.

```
cd /etc/apache2/sites-available
ed default <<'%'
g;RailsBaseURI /apps/contacts;d
$i
    <Location "/contacts/">
        Order deny,allow
        Deny from all
    </Location>
.
w
q
%
/etc/init.d/apache2 stop
/etc/init.d/apache2 start
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones
http://www.abcd.ox.ac.uk:8110/apps/contacts/phones
```

4.3 Getting Rails to ask for a username and password

To ensure that the visitor has authenticated before any method of the controller is executed, we alter the controller so that its first line of code is a call of `before_filter`. This call ensures that, if a visitor tries to call any of the methods (apart from the method called `index`), a method called `authenticate` will get called first.

Note: we've put the `authenticate` method in the private part of the class declaration. This means that it cannot be called directly by a visitor, only indirectly by a method of this class.

In the body of the `authenticate` method, there is a call of a method that has the wonderfully long name `authenticate_or_request_with_http_basic`. If the visitor has not yet authenticated, then this method arranges for the visitor's browser to display a login box asking the visitor to type in a username and password. If, instead, the visitor authenticated earlier, the visitor's browser will secretly pass the username and password entered earlier to the server.

The `authenticate` method can choose which usernames and passwords to accept. In the example below, the code is checking whether they have the values `UN4login` and `PW4login`. In a real example, the username and password might be looked up in a database or an LDAP server.

If the username and password are acceptable, `authenticate_or_request_with_http_basic` will allow the Rails application to continue; otherwise, it will return an error message to the user's browser.

```
cd /var/apps/contacts
ed app/controllers/phones_controller.rb <<%
/class PhonesCon/a
  before_filter :authenticate, :except => [ :index ]
.
/^end/i
  private
  def authenticate
    authenticate_or_request_with_http_basic do |user, pass|
      user=="UN4login" && pass=="PW4login"
    end
  end
.
w
q
%
/etc/init.d/apache2 stop
/etc/init.d/apache2 start
```

We can test this using the following two URLs. Because of the `:except` parameter to the `before_filter` method call, the first URL does not require authentication. However, the second URL will display a login box requiring us to provide the appropriate username and password.

```
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones
https://www.abcd.ox.ac.uk:8113/apps/contacts/phones/new
```