Integrate Lighttpd and Mongrel clusters with IIS

Rails applications usually run pretty well. On Windows machines, applications tend to run a little slower than other operating systems mainly because of the performance of Ruby.

Because Rails is not thread-safe, requests are handled one at a time. This means that applications that allow users to make long requests could block other users. Let's say we have an application that allows a user to upload a movie and the upload takes 5 minutes. Other users will not be able to do anything with the application during that period. The easy solution to this is to add more dispatchers into the mix.

This article will demonstrate how to set up Lighttpd on Windows and have it forward requests to a cluster of Mongrel services. We'll then go about putting that behind IIS.

We'll even integrate it with an existing website by having it mounted to a virtual folder (eg: http://myserver.mydomain.com/test/).

Assumptions

This article assumes that you have a working Rails application to test, that you are familiar with how IIS works. IIS should be running on TCP Port 80 (or 443 if you're doing SSL).

Shopping List

In order to make this work, you'll need to download the full version of ISAPI Rewrite from a company called Helicon. You can obtain a free 30 day unlimited trial from their web site but they charge $70 per server (or less if you buy more than once license) if you want to use it in production. While there are free rewrite plugins available, this is the only one I know of that provides proxy capabilities for IIS. Visit http://www.isapirewrite.com/ for more information.

You'll also want to grab the Windows 2003 Server Resource Kit (http://www.microsoft.com/downloads/details.aspx?familyid=9d467a69-57ff-4ae7-96ee-b18c4790cffd&displaylang=en) from Microsoft If you want to set up Lighttpd as a Windows Service.

Installing Lighttpd for Windows

Kevin Worthington maintains a Windows build of Lighttpd on his web site. At the time of writing, the current version is 1.4.11. You can obtain a copy here: http://www.kevinworthington.com:8181/?p=116

Install Lighttpd, accepting all defaults. It should install to **c:\lighttpd**.

Installing Mongrel

1. Open a command prompt
2. Install mongrel and dependencies (gem install mongrel –include-dependencies)
3. Choose the **win32** option!

Set up a new Rails app

1. Navigate to your **c:\web** folder
   cd\web

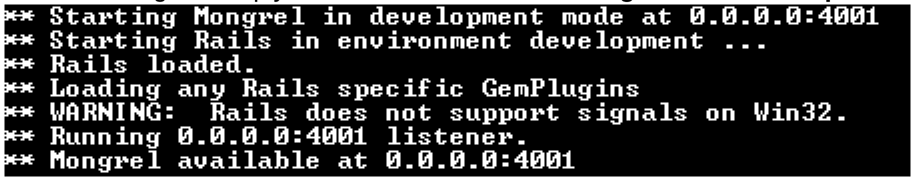
2. Create a new Rails application in that folder (or add your own **working** application)
   rails app2

3. Ensure the Rails application works by testing it with WEBrick. Make sure that the database configuration for production is correct. (Sorry, that's not covered here!)
   cd\web\app2
   ruby script/server –e production –p 4001

4. Stop WEBrick with CTRL+Break

Test with Mongrel

To test with Mongrel, simply execute the command **mongrel_rails start –p 4001.**



```
** Starting Mongrel in development mode at 0.0.0.0:4001
** Starting Rails in environment development ...
** Rails loaded.
** Loading any Rails specific GemPlugins
** WARNING:  Rails does not support signals on Win32.
** Running 0.0.0.0:4001 listener.
** Mongrel available at 0.0.0.0:4001
```

If you see that, you're good to go! You can now install this application as a service in Production mode!. Of course, you should test it by navigating to http://localhost:4001/

Set up the Mongrel cluster

On Linux, we'd use the awesome mongrel_cluster gem. Unfortunately it won't work for us on Windows because of bugs in the way Ruby works and the fact that Mongrel simply cannot run as a daemon.

So we'll just install two instances of the same app as Windows serivces.

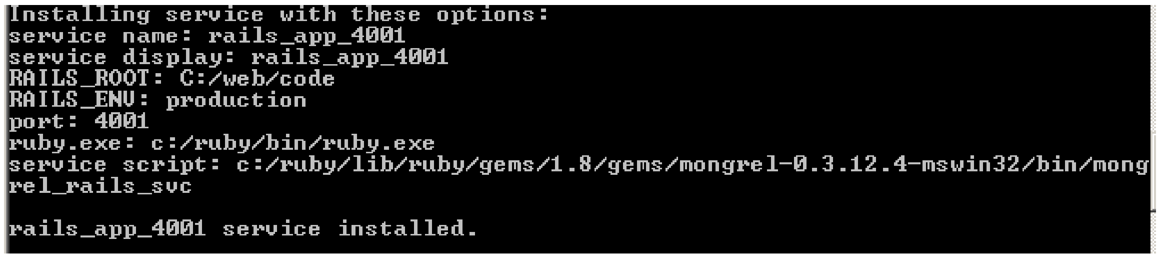
Install Mongrel as a Windows Service

We'll install this application using production mode, so make sure your **database.yml** file points to a working production database if you're using your own application with this tutorial.

1. Make sure Mongrel is stopped by pressing **ctrl+break**. Answer "Yes" to "Terminate Batch Job" if you are prompted.

2. Execute the command below to install the application as a service

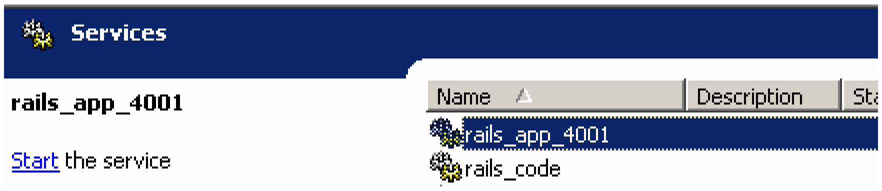
   mongrel_rails_service install –n rails_app_4001 –p 4001

```
Installing service with these options:
service name: rails_app_4001
service display: rails_app_4001
RAILS_ROOT: C:/web/code
RAILS_ENV: production
port: 4001
ruby.exe: c:/ruby/bin/ruby.exe
service script: c:/ruby/lib/ruby/gems/1.8/gems/mongrel-0.3.12.4-mswin32/bin/mong
rel_rails_svc

rails_app_4001 service installed.
```

   This will create a new Windows service with the name **rails_app_4001** which you can view in the Control Panel.



3. You can start the service from Control Panel or you can start it from the command line by executing the command **mongrel_rails_service start –n rails_app_4001**
    a. To stop the service, you can use **mongrel_rails_service stop –n rails_app_4001**
    b. If something didn't work right, you can remove the service **mongrel_rails_service delete –n rails_app_4001**

4. You'll want to set the startup type of the service to **automatic** when you're done so that the service will start when the machine restarts.

5. Install another service that points to the same application, this time on port 4002 and start that service
   mongrel_rails_service install –n rails_app_4002 –p 4002
   mongrel_rails_service start –n rails_app_4002

6. Test each address to make sure that the requests work and that the services are in fact serving your web applications.

Configure Lighttpd for load balancing

1. Open the file **c:\lighttpd\etc\lighttpd.conf** and change the default port from 80 to something higher like 81.

```
server.port = 80
```

2. Ensure the following server modules are loaded:

```
server.modules  =  ("mod_proxy",
                     "mod_rewrite",
                      "mod_accesslog",
                      "mod_alias" )
```

3. Make sure you have a default document location

```
server.document-root      = "c:/lighttpd/htdocs"
server.indexfiles = ( "index.htm", "index.html" )
```

4. At the bottom of the file, place the proxy directives

```
proxy.debug = 0
 proxy.balance = "fair"
 proxy.server  = ( "/" =>
    (
      ( "host" => "127.0.0.1", "port" => 4001 ),
      ( "host" => "127.0.0.1", "port" => 4002 )
    )
 )
```

5. Save the file. Refer to the appendix for the complete configuration file if you get stuck.
6. Start Lighttpd. From the command line, type **c:\lighttpd\sbin\lighttpd.exe -D -f c:\lighttpd\etc\lighttpd.conf**

IIS Integration

Installl ISAPI Rewrite

Visit http://www.isapirewrite.com/ and download the trial version of the ISAPI Rewrite plugin.

- Direct download is http://www.isapirewrite.com/download/isapi_rwf_x86_0060.msi
- Launch the installation program and accept all of the default settings.
- You will be prompted to restart IIS and you should allow this.
- If you experience trouble with the installation, you'll need to refer to the developers of this product.

Fixing the ISAPIRewrite association issue

After installing, it may be necessary to "fix" the association of this filter.

- Open a command prompt
- Navigate to **C:\Program Files\Helicon\ISAPI_Rewrite**
- Launch the ProxyCfg.vbs script
  - Proxycfg.vbs –r or cscript proxycfg.vbs –r
- Restart IIS

Configure ISAPI Rewrite

The last step is to modfy the httpd.ini file which resides in **C:\Program Files\Helicon\ISAPI_Rewrite**

Add this to the bottom of the file.

```
# FOR TEST APPLICATION forwarding to lighttpd
# on port 81
RewriteProxy /test(.*) http\://localhost:81$1 [I,U]
```

Save the file. You should not need to restart IIS.

Testing it out!

If all worked well, you can now pull up your Rails application via IIS by navigating to http://localhost/test/

Unfortunately, it's not going to look very good. Read on to find out why.

Reverse Proxy and URLs

The big problem we're faced with now is that the URLs that Rails creates internally, such as stylesheet links, url_for links and other links don't work as we expect… instead, they direct users around the proxy. This is bad because it exposes the proxied server.

IIS has no method to handle reverse proxying. A reverse proxy rewrites the content served from the backend to mask the fact that the request was filtered through a proxy.

Thankfully, there's a way around this… using a simple Rails plugin that modifieds the way Rails creates its URLs. We're going to make Rails prepend our external URL to any URLs it creates through the system. This will force all user requests to come back through the IIS proxy.

Installing the proxy plugin

Execute the command

```
ruby script/plugin install http://svn.napcsweb.com/public/reverse_proxy_fix
```

from within your application's root folder. The plugin should install and then ask you for the base url. Enter **http://localhost/test** and pres 'enter'. If all goes well, the configuration file will be written. If the configuration file can't be modified, you can navigate to **vendor/plugins/reverse_proxy_fix** and change it yourself.

If the installation fails, you can build the plugin yourself if you follow the next section.

Creating the proxy plugin

If you don't have Subversion installed, you can follow these steps to get the plugin configured properly.

- Create a new Rails plugin called "reverse_proxy_fix"
  ruby/script generate reverse_proxy_fix
- Navigate to your application's vendor/plugins/reverse_proxy_fix folder and edit the init.rb file
  - Add the following code to the file
    Require 'reverse_proxy_fix"
- Edit **vendor/plugins/iis_proxy_fix/lib/reverse_proxy_fix.rb** and replace the contents with the code located in the appendix.
- Modify the first line to match your application's url…
- **BASE_URL = 'http://localhost/app1'**

Using the proxy plugin

Once the plugin is installed, you'll need to restart your Rails application. In this case, you need to restart both Mongrel services for the changes to take effect. You do not need to restart Lighttpd.

If all went as expected, any internal links in your application should now be corrected and your users will routed back through the proxy.

The plugin is only active in production mode, so it's safe to keep in your application during the development process.

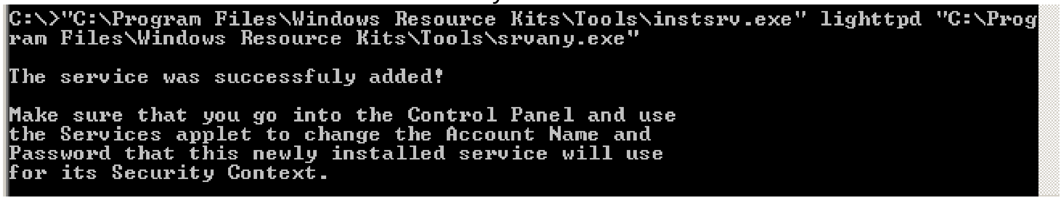
Making Lighttpd run as a service

1. Download the Windows 2003 Server Resource Kit from Microsoft and install it. (http://www.microsoft.com/downloads/details.aspx?familyid=9d467a69-57ff-4ae7-96ee-b18c4790cffd&displaylang=en

2. Install the file and choose the defaults

3. Open a command prompt and enter the following command:

   "C:\Program Files\Windows Resource Kits\Tools\instsrv.exe" lighttpd "C:\Program Files\Windows Resource Kits\Tools\srvany.exe"



```
C:\>"C:\Program Files\Windows Resource Kits\Tools\instsrv.exe" lighttpd "C:\Prog
ram Files\Windows Resource Kits\Tools\srvany.exe"

The service was successfuly added!

Make sure that you go into the Control Panel and use
the Services applet to change the Account Name and
Password that this newly installed service will use
for its Security Context.
```

4. Open **regedit** and locate the key **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\lighttpd**

5. Create a new key beneath that key called **Parameters**

6. Select the **Parameters** key and create a new **String Value** with the name **Application**

7. Enter the following for the value
   c:\lighttpd\sbin\lighttpd.exe -D -f c:\lighttpd\etc\lighttpd.conf

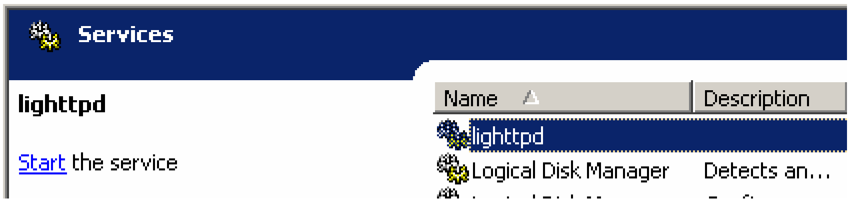
8. Ensure that all lighttpd instances are stopped by executing the following command at the command line
   c:\lighttpd\sbin\process.exe -k lighttpd.exe

9. Start the service by typing **net start lighttpd**

10. Stop the service by typing **net stop lighttpd**

11. Close the Registry Editor

Your new service should be viewable from the Services control panel.



Test your application through the browser at http://localhost:81/ to ensure that Lighttpd is working as a proxy.

Finally, test your application through IIS to make sure that the entire chain works as you expect, including the URL rewriting.

Wrapping Up

Your application is now set up to scale to more instances of Mongrel or more machines. Do some performance testing with **httperf** and see what you get for requests per second. Then figure out how many more Mongrel services you need to add to handle the load. Using Lighttpd as your load balancer, you can add Mongrel services as you need them and they can even be on other machines.

Acknowledgements

Kevin Worthington (Lighttpd for Windows)
Zed Shaw (Mongrel)
Chris Hulbert (Original instructions for installing Lighttpd as a service)
Bradley Taylor (Mongrel_cluster)

Appendix

Lighttpd config file

```
# Default configuration file for the lighttpd web server
# Start using ./script/server lighttpd

server.port = 81
server.modules          = ("mod_proxy",  "mod_rewrite", "mod_accesslog" )

server.document-root     = "c:/lighttpd/htdocs"
server.indexfiles = ( "index.htm", "index.html" )

 proxy.debug = 0
 proxy.balance = "fair"
 proxy.server  = ( "/" =>
     (
       ( "host" => "127.0.0.1", "port" => 4000 ),
      ( "host" => "127.0.0.1", "port" => 4001 )
     )
  )
```

ISAPI Rewrite httpd.ini file

```
[ISAPI_Rewrite]
# 3600 = 1 hour
CacheClockRate 3600

RepeatLimit 32

# Block external access to the httpd.ini and httpd.parse.errors files
RewriteRule /httpd(?:\.ini|\.parse\.errors).* / [F,I,O]
# Block external access to the Helper ISAPI Extension
RewriteRule .*\.isrwhlp / [F,I,O]

# Proxy requests to Apache on 8080.

# TEST APPLICATION
RewriteProxy /test(.*) http\://localhost:81$1 [I,U]
```

plugin/reverse_proxy_fix/lib/reverse_proxy_fix.rb

```
# Copyright © 2006 Brian Hogan
#
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction, including
# without limitation the rights to use, copy, modify, merge, publish,
# distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so, subject to
# the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
# LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
# WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

BASE_URL = ‘’
module ActionController

  protected
  # Configure the prefix on the url only if we're running in production mode
  # Throws an exception if the BASE_URL constant has not been configured in
  # config.rb
  def self.check_mode_and_base
        if RAILS_ENV == 'production'
                return true
        else
                return false
        end
  end

  # Set the asset host for CSS, JS, and image files if we're in production
  # mode and the base_path has been configured.
  if check_mode_and_base
        ActionController::Base.asset_host = BASE_URL
  end

  # Modifies the original UrlRewriter class, altering how the URLs are created.
  class UrlRewriter

        alias old_rewrite_url rewrite_url

        # Prepends the BASE_URL to all of the URL requests created by the
        # URL rewriter in Rails, stripping off the host, port, etc to ensure that
        # the new URL is exactly what you expect.
        #
        # This method calls check_mode_and_base to ensure that the URL fixing only occurs
        # in production mode and that the BASE_URL variable in config.rb is set.
        def rewrite_url(path, options)
                url = old_rewrite_url(path, options)
                url = url.gsub(@request.protocol + @request.host_with_port, '')
                if ActionController::check_mode_and_base
                        url = BASE_URL + url
                end
                url

        end
  end

end
```