Getting More Out Of Apache (Part 1)

By icarus

Table of Contents

Genesis

Of the diverse open−source applications currently available, perhaps the most impressive success stories are those of the Linux kernel and the Apache Web server. The story behind the Linux kernel is a well−known one − the tireless efforts of Linus Torvalds and his team of developers to create an open, stable and scalable operating system − but perhaps less well−known is the story of how the Apache project began, and how it has grown to the point where 60% of the Web runs on Apache.

The Apache project began in 1995, as a collaborative effort between a group of webmasters who wanted to build a "robust and commercial−grade implementation of the HTTP protocol" (Apache.org), and make this available to the user community absolutely free of charge. Originally conceived as a series of patches to the original NCSA httpd daemon, the project ultimately took on a life of its own, with the NCSA daemon undergoing several redesigns in order to make it more extensible and modular.

The end result (perhaps that's not the right phrase, since the Apache server is continuously evolving) is a Web server that's powerful enough to meet the needs of both large and small Web sites, yet simple enough to configure that you can get it up and running on your UNIX (or Windows) system in less than five minutes.

The popularity of Apache among Web site administrators and developers can be judged from recent Netcraft findings, which reveal that 15,414,726 Web sites, or a little over 60% of the total Web sites on the Internet, run on Apache (http://www.netcraft.com/survey/). Pretty impressive for a program that began life as a series of patches (incidentally, the term "Apache Server" is derived from the words "A PAtCHy Server" − a bow to Apache's genesis as a series of patches applied to an existing httpd daemon)

Now, while the out−of−the−box Apache configuration is usually more than satisfactory for those without special requirements, Apache does allow you to customize its behaviour extensively. Most of this customization takes place via the Apache configuration file, "httpd.conf", and is implemented via directives in this file.

Across this article and the next, I'll be discussing some of the more interesting things you can do with Apache, both to help you maximize your usage of the Web server and to increase your familiarity with some of the features hidden under the hood of this wonderful piece of software.

Virtually Yours

It's a common misconception that a single Web server can host, or "serve", only a single Web site, since a server typically has only a single IP address assigned to it. This is not at all true − and over the next couple of pages, I'll be showing you how you can use Apache to serve up more than one Web site at a time.

First, though, the basics − every computer on the Internet is assigned an IP address, which serves as a unique identifier for that computer. In order to connect to a Web site, a user needs to know the IP address of the computer on which that Web site is stored. However, since users cannot be expected to remember strings of numbers for different Web sites, the Domain Name System was introduced to make things a little easier.

The Domain Name System, or DNS, maps each numeric string to an easy−to−remember word or phrase. For example, the IP address 216.115.108.243 and the word "yahoo.com" both refer to the same Web site; however, the latter is much easier to remember. By mapping names to IP addresses, the DNS makes it easier to navigate the Web.

What does this have to do with Apache? Well, Apache was the first Web server to introduce "name−based virtual hosting". The concept is simple and elegant: different domain names all point to the same IP address, and the Web server at that IP address has the intelligence necessary to display a different Web page for each domain.

Name−based virtual hosting has been available since the HTTP/1.1 protocol came out. However, there is one gotcha − in order for this to work, the client browser must also include support for HTTP/1.1. Most newer browsers do include this support − and for those which don't, there's a workaround which allows them to perceive similar functionality.

Alpha And Beta

The keys to name−based virtual hosting are the appropriately−named NameVirtualHost and directives in the "httpd.conf" configuration file; these directives are used to specify basic information for the "virtual host", such as server name and server root, administrator email address, and log file locations.

Let's consider a simple scenario − creating two virtual hosts on the domain "localhost" (your Linux box) for the Web sites "melonfire−alpha.com" and "melonfire−beta.com". With the new NameVirtualHost directive, this is simplicity itself − open up the "httpd.conf" file in your favourite text editor, look for the "Virtual Hosts" section, and add the following entry to it:

```
NameVirtualHost 127.0.0.1:80
```

The NameVirtualHost directive specifies the IP address of the server which will be used to resolve virtual host names.

With that out of the way, it's now time to begin adding the virtual host definitions themselves.

```
<VirtualHost 127.0.0.1> ServerAdmin
webmaster@melonfire−alpha.com DocumentRoot
/www/melonfire−alpha.com ServerName melonfire−alpha.com
ErrorLog logs/melonfire−alpha.com−error_log CustomLog
logs/melonfire−alpha.com−access_log common </VirtualHost>
```

Let's dissect this a little.

The first line

```
<VirtualHost 127.0.0.1>
```

specifies the IP address of the virtual host. Very simply, this is the IP address of the machine that holds the Web pages you plan to display. In the example above, the machine is "localhost", which traditionally has the IP address 127.0.0.1

The second line

```
ServerAdmin webmaster@melonfire−alpha.com
```

specifies the email address of the administrator for this virtual host.

The third and fourth lines

```
DocumentRoot /www/melonfire−alpha.com ServerName
melonfire−alpha.com
```

are probably the most important − they specify the domain name for the virtual host, and the physical location of the Web pages for that domain on the hard drive. Both these lines are crucial to ensuring that the correct Web page appears when the domain is accessed by a client browser.

Finally, the remaining two lines

```
ErrorLog logs/melonfire−alpha.com−error_log CustomLog
logs/melonfire−alpha.com−access_log common
```

specify the files to which server activity is to be logged. You can log server visits and server errors to separate files for each of your virtual hosts, or put it all into the common log files (the default option).

You can also add other Apache directives to this virtual host entry. Once you're done, close the entry with a

```
</VirtualHost>
```

tag.

Two Birds With One Host

Here's what the final product looks like:

```
# host setting for melonfire−alpha.com <VirtualHost 127.0.0.1>
ServerAdmin webmaster@melonfire−alpha.com DocumentRoot
/www/melonfire−alpha.com ServerName melonfire−alpha.com
ErrorLog logs/melonfire−alpha.com−error_log CustomLog
logs/melonfire−alpha.com−access_log common </VirtualHost> #
host setting for melonfire−beta.com <VirtualHost 127.0.0.1>
ServerAdmin webmaster@melonfire−beta.com DocumentRoot
/www/melonfire−beta.com ServerName melonfire−beta.com ErrorLog
logs/melonfire−beta.com−error_log CustomLog
logs/melonfire−beta.com−access_log common </VirtualHost>
```

And you can test it by restarting the Apache server and pointing your browser to the Web sites http://melonfire−alpha.com and http://melonfire−beta.com (note that you may need to update your DNS tables as well for this to work).

On the assumption that you have created HTML files for each host and stored them in the locations you've specified for the DocumentRoot directive, Apache should display the appropriate index page for each server. And if you take a look at the log files, you should see separate log files for each virtual host.

As you might imagine, the ability to host multiple Web sites on a single server is an elegant − and economical − solution to the problem of limited IP address range in the IPv4 protocol. It also allows Web hosting services to "sub−host" multiple client domains on a single server, and developers to run multiple Web sites on their development and test systems for simulation purposes.

Gotcha!

There are, however, a couple of gotchas with virtual hosting.

Once you have specified the NameVirtualHost directive, it's important to remember that all the other default settings in your configuration file are null and void. Consequently, if you wish to access the default server "localhost", you will need to include it in the list of virtual hosts.

The second gotcha is the one mentioned a few pages back − clients which do not support HTTP/1.1 will not be able to view virtual hosts correctly. The Apache documentation addresses this problem via the ServerPath directive.

Here's what it would look like:

```
# host setting for melonfire−alpha.com <VirtualHost 127.0.0.1>
ServerAdmin webmaster@melonfire−alpha.com DocumentRoot
/www/melonfire−alpha.com ServerName melonfire−alpha.com
ServerPath /alpha ErrorLog logs/melonfire−alpha.com−error_log
CustomLog logs/melonfire−alpha.com−access_log common
</VirtualHost> # host setting for melonfire−beta.com
<VirtualHost 127.0.0.1> ServerAdmin
webmaster@melonfire−beta.com DocumentRoot
/www/melonfire−beta.com ServerName melonfire−beta.com
ServerPath /beta ErrorLog logs/melonfire−beta.com−error_log
CustomLog logs/melonfire−beta.com−access_log common
</VirtualHost>
```

How does this work? Very simply − any requests for URLs beginning with the descriptor /alpha will be automatically served from host "melonfire−alpha.com", while URLs beginning with the descriptor /beta will be served from host "melonfire−beta.com".

For more information on this, you should take a look at the Apache documentation, which includes a working example.

Including A Few More Tricks

The next item on today's agenda involves something called "server−side includes", or SSI.

Very simply, server−side includes are sets of instructions which enable Web servers to display data contained in server−side variables (server date and time, file currently in use, and so on). SSI also allows you to include external text files in your HTML pages − this comes in handy when you need to add a common header and footer to every HTML page.

Adding support for server−side includes is again a matter of tweaking the Apache configuration file, "httpd.conf". By default, the server root is not configured to parse documents containing SSI instructions. In order to enable this support, you need to add the

```
Includes
```

keyword to the <Directory> directive − it should look something like this:

```
Options Indexes FollowSymLinks Includes
```

Next, you need to tell the server to recognize SSI files − these typically have the file extension ".shtml". In order to do this, add the following lines to the configuration file:

```
# specify MIME type for SSI files AddType text/html .shtml
AddHandler server-parsed .shtml
```

Restart Apache for the changes to take effect.

Hello, World!

With that done, it's time to take see how server−parsed HTML files work. Create the following simple file, and save it as "hello.shtml".

```
<html> <head> </head> <body> You have just loaded <!−−#echo
var="DOCUMENT_NAME" −−> </body> </html>
```

Now, when you point your browser to this page (via the Web server), you should see something like this

```
You have just loaded hello.shtml
```

Thus, the SSI directive allows you to obtain the name of the current document without you − the document author − having to explicitly specify it.

SSI also allows you to display the size and last modification time of the current file − take a look!

```
<html> <head> </head> <body> You are currently viewing <!−−#echo
var="DOCUMENT_NAME" −−> (<!−−#fsize file="$DOCUMENT_NAME" −−>), last
modified on <!−−#flastmod file="$DOCUMENT_NAME"−−> </body> </html>
```

And the output would be:

```
You are currently viewing hello.shtml ( 1k), last modified on Thursday, 04−Jan−2001
21:28:33 IST
```

And you can also display the current server time, as illustrated below:

```
<!−−#config timefmt="%I:%M %p" −−> <html> <body> Current time is <!−−#echo
var="DATE_LOCAL" −−> </body> </html>
```

Finally, SSI also allows you to separate common elements of your Web pages into separate files, and include these files on each and every page. For example, if I have a copyright notice that is to be displayed on each and every page, I could place it in a file named "footer.txt"

```
This material copyright Melonfire, 2001. All rights reserved.
```

and then create HTML pages which looked like this:

```
<html> <head> </head> <body> This is where proprietary material goes. <p> <!−−#include
file="footer.txt"> </body> </html>
```

Now, each time a page like the one above is requested from the Web server, Apache will automatically include the contents of the file "footer.txt" in the appropriate place. The final page might look something like this:

This is where proprietary material goes. This material copyright Melonfire, 2001. All rights reserved.

Be warned, however, that extensive use of SSI can affect Web server performance, since the server has to first parse the file before sending it to the client browser − so use it judiciously.

Detailed information on SSI can be obtained from the SSI resources at http://httpd.apache.org/docs/howto/ssi.html and http://www.echodev.com/tutorials/ssi/basics/

And that's about it for this time. Come back for the second part of this article, where I'll be discussing URL rewriting, user authentication and a few other tricks.