
Virtual Hosts

As a person can be known by many names, so can a Web server support multiple Web sites. In the Apache configuration file, each alternate identity, and probably the “main” one as well, is known as a virtual host (sometimes written as *vhost*) identified with a `<VirtualHost>` container directive. Depending on the name used to access the Web server, Apache responds appropriately, just as someone might answer differently depending on whether she is addressed as “Miss Jones” or “Hey, Debbie!” If you want to have a single system support multiple Web sites, you must configure Apache appropriately—and you’ll need to know a little bit about your system (such as the IP addresses assigned to it) in order to do it correctly.

There are two different types of virtual host supported by Apache. The first type, called address-based or IP-based, is tied to the numeric network address used to reach the system, rather like telephone numbers. Bruce Wayne never answered the parlour telephone with “Batman here!” nor did he answer the phone in the Batcave by saying, “Bruce Wayne speaking.” However, it’s the same person answering the phone, just as it’s the same Web server receiving the request. Even if the caller had a wrong number and said, “Hi, Steve!,” the phone was still answered the same way; nothing would convince Batman to admit on the Batphone that it was Bruce Wayne answering.

The other type of virtual host is called name-based, because the server’s response depends on the name by which it was called. To continue the telephone analogy, consider an apartment shared by multiple roommates; you call the same number whether you want to speak to Dave, Joyce, Amaterasu, or Georg. Just as multiple people may share a single telephone number, multiple Web sites can share the same IP address. However, all IP addresses shared by multiple Apache virtual hosts need to be declared with a *NameVirtualHost* directive.

In the most simple of Apache configurations, there are no virtual hosts. Instead, all of the directives in the configuration file apply universally to the operation of the server. The environment defined by the directives outside any `<VirtualHost>` containers is sometimes called the “default server,” “main server,” or perhaps the “global server.”

There is no official name for it, but it can become a factor when adding virtual hosts to your configuration.

But what happens if you add a *<VirtualHost>* container to such a configuration? How are those directives outside the container interpreted, and what is their effect on the virtual host?

The answer is not a simple one: essentially, the effect is specific to each configuration directive. Some get inherited by the virtual hosts, some get reset to a default value, and some pretend they've never been used before. You'll need to consult the documentation for each directive to know for sure.

There are two primary forms of virtual hosts: IP-based virtual hosts, where each virtual host has its own unique IP address; and name-based virtual hosts, where more than one virtual host runs on the same IP address but with different names. This chapter will show you how to configure each one and how to combine the two on the same server. You'll also learn how to fix common problems that occur with virtual hosts.



To avoid problems and confusing error messages, we strongly advise that you explicitly include the port number on directives specifying an IP address, if they support supplying both the address and the port. For instance, use

```
NameVirtualHost *:80
```

instead of

```
NameVirtualHost *
```

Normal Web operations use port 80, and most SSL requests use port 443.

4.1 Setting Up Name-Based Virtual Hosts

Problem

You have only one IP address, but you want to support more than one Web site on your system.

Solution

Use the *NameVirtualHost *:80* directive in conjunction with *<VirtualHost>* sections:

```
ServerName 127.0.0.1
NameVirtualHost *:80


<VirtualHost *:80>
    ServerName TheSmiths.name
    DocumentRoot "C:/Apache/Sites/TheSmiths"
</VirtualHost>
```

```
<VirtualHost *:80>
    ServerName JohnSmith.name
    DocumentRoot "C:/Apache/Sites/JustJohnSmith"
</VirtualHost>
```

Discussion

With IP addresses increasingly hard to come by, name-based virtual hosting is the most common way to run multiple Web sites on the same Apache server. The previous recipe works, for most users, in most virtual hosting situations.

The `*:80` in the previous rules means that the specified hosts run on all addresses. For a machine with only a single address, this means that it runs on that address but will also run on the *loopback*, or *localhost* address. Thus if you are sitting at the physical server system, you can view the Web site.

The argument to the `<VirtualHost>` container directive needs to match the argument in a `NameVirtualHost` directive. Putting the hostname here may cause Apache to ignore the virtual host on server startup, and requests to this virtual host may unexpectedly go somewhere else. If your name server is down or otherwise unresponsive at the time that your Apache server is starting up, then Apache can't match the particular `<VirtualHost>` section to the `NameVirtualHost` directive to which it belongs. 

Requests for which there is not a virtual host listed will go to the first virtual host listed in the configuration file. In the case of the previous example, requests coming to the server using hostnames that are not explicitly mentioned in one of the virtual hosts will be served by the `TheSmiths.name` virtual host.

It is particularly instructive to run `httpd -S` and observe the virtual host configuration as Apache understands it, to see if it matches the way that you understand it. `httpd -S` returns the virtual host configuration, showing which hosts are name-based, which are IP-based, and what the defaults are.

Multiple names can be listed for a particular virtual host using the `ServerAlias` directive, as shown here:

```
ServerName TheSmiths.name
ServerAlias www.TheSmiths.name Smith.Family.name
```

It is important to understand that virtual hosts render the server listed in the main body of your configuration file (the “main” or “default” server mentioned earlier) no longer accessible—you must create a virtual host section explicitly for that host. List this host first, if you want it to be the default one.

Adding name-based virtual hosts to your Apache configuration does not magically add entries to your DNS server. You must still add records to your DNS server so that the names resolve to the IP address of the server system. When users type your server name (s) into their browser location bars, their computers first contact a DNS server to look

up that name and resolve it to an IP address. If there is no DNS record, then their browsers can't find your server.

For more information on configuring your DNS server, consult the documentation for the DNS software you happen to be running, or talk to your ISP if you're not running your own DNS server.

See Also

- <http://httpd.apache.org/docs/2.2/vhosts/>

4.2 Designating One Name-Based Virtual Host as the Default

Problem

You want all unmatched requests, whether they specify a name or use an IP address, to be directed to a default host, possibly with a “host not found” error message.

Solution

Add the following `<VirtualHost>` section, and list it before all of your other ones:

```
<VirtualHost *:80>
    ServerName default
    DocumentRoot /www/htdocs
    ErrorDocument 404 /site_list.html
</VirtualHost>
```

Discussion

Note that this recipe is used in the context of name-based virtual hosts, so it is assumed that you have other virtual hosts that are also using the `<VirtualHost *:80>` notation, and that there is also an accompanying `NameVirtualHost *:80` appearing above them. We have used the `default` name for clarity; you can call it whatever you want.

Setting the `ErrorDocument 404` to a list of the available sites on the server directs the user to useful content, rather than leaving him stranded with an unhelpful 404 error message. You may wish to set `DirectoryIndex` to the site list as well, so that users who go directly to the front page of this site also get useful information.

It's a good idea to list explicitly all valid hostnames either as `ServerNames` or `ServerAliases`, so that nobody ever winds up at the default site. However, if someone accesses the site directly by IP address, or if a hostname is added to the address in question before the appropriate virtual host is created, the user still gets useful content.

See Also

- Recipe 4.4

4.3 Setting Up Address-Based Virtual Hosts

Problem

You have multiple IP addresses assigned to your system, and you want to support one Web site on each.

Solution

Create a virtual host section for each IP address you want to list on:

```
ServerName 127.0.0.1

<VirtualHost 10.0.0.1>
    ServerName Example.Com
    DocumentRoot "C:/Apache/Sites/Example.Com"
</VirtualHost>

<VirtualHost 10.0.0.2>
    ServerName JohnSmith.Example.Com
    DocumentRoot "C:/Apache/Sites/JustJohnSmith"
</VirtualHost>
```

Discussion

The virtual hosts defined in this example catch all requests to the specified IP addresses, regardless of what hostname is used to get there. Requests to any other IP address not listed go to the virtual host listed in the main body of the configuration file.

The *ServerName* specified is used as the primary name of the virtual host, when needed, but is not used in the process of mapping a request to the correct host. Only the IP address is consulted to figure out which virtual host to serve requests from, not the *Host* header field.

See Also

- <http://httpd.apache.org/docs/2.2/vhosts/>

4.4 Creating a Default Address-Based Virtual Host

Problem

You want to create a virtual host to catch all requests that don't map to one of your address-based virtual hosts.

Solution

Use the *_default_* keyword to designate a default host:

```
<VirtualHost _default_>
    DocumentRoot /www/htdocs
</VirtualHost>
```

Discussion

The `_default_` keyword creates a virtual host that catches all requests for any `address:port` combinations for which there is no virtual host configured.

The `_default_` directive may—and should—be used in conjunction with a particular port number, such as:

```
<VirtualHost _default_:443>
```

Using this syntax means that the specified virtual host catches all requests to port 443, on all addresses for which there is not an explicit virtual host configured. SSL virtual hosts are usually set up using the `_default_` syntax, so you'll see this syntax used in the default SSL configuration file, along with the necessary directives to enable SSL.

`_default_` typically does not work as people expect in the case of name-based virtual hosts. It does not match names for which there are no virtual host sections, only `address:port` combinations for which there are no virtual hosts configured. If you wish to create a default name-based host, see Recipe 4.2.

See Also

- Recipe 4.2

4.5 Mixing Address-Based and Name-Based Virtual Hosts

Problem

You have multiple IP addresses assigned to your system, and you want to support more than one Web site on each address.

Solution

Provide a *NameVirtualHost* directive for each IP address, and proceed as you did with a single IP address:

```
ServerName 127.0.0.1
NameVirtualHost 10.0.0.1:80
NameVirtualHost 10.0.0.2:80

<VirtualHost 10.0.0.1:80>
    ServerName TheSmiths.name
    DocumentRoot "C:/Apache/Sites/TheSmiths"
</VirtualHost>

<VirtualHost 10.0.0.1:80>
    ServerName JohnSmith.name
```

```

        DocumentRoot "C:/Apache/Sites/JustJohnSmith"
    </VirtualHost>

    <VirtualHost 10.0.0.2:80>
        ServerName Example.Com
        DocumentRoot "C:/Apache/Sites/Example.Com"
    </VirtualHost>

    <VirtualHost 10.0.0.2:80>
        ServerName Doriferguson.Example.Com
        DocumentRoot "C:/Apache/Sites/JustDoriferguson"
    </VirtualHost>

```

Discussion

Using the address of the server, rather than the wildcard `*` argument, makes the virtual hosts listen only to that IP address. However, you should notice that the argument to `<VirtualHost>` still must match the argument to the `NameVirtualHost` with which they are connected.

See Also

- <http://httpd.apache.org/docs/2.2/vhosts/>

4.6 Mass Virtual Hosting with mod_vhost_alias

Problem

You want to host many virtual hosts, all of which have exactly the same configuration.

Solution

Use `VirtualDocumentRoot` and `VirtualScriptAlias` provided by `mod_vhost_alias`.

```

VirtualDocumentRoot /www/vhosts/%-1/%-2.1/%-2/htdocs
VirtualScriptAlias   /www/vhosts/%-1/%-2.1/%-2/cgi-bin

```

Discussion

This recipe uses directives from `mod_vhost_alias`, which you may not have installed when you built Apache, as it is not one of the modules that is enabled by default.

These directives map requests to a directory built up from pieces of the hostname that was requested. Each of the variables represents one part of the hostname, so that each hostname is mapped to a different directory.

In this particular example, requests for content from `www.example.com` are served from the directory `/www/vhosts/com/e/example/htdocs`, or from `/www/vhosts/com/e/example/cgi-bin` (for CGI requests). The full range of available variables is shown in Table 4-1.

Table 4-1. *mod_vhost_alias* variables

Variable	Meaning
%%	insert a %
%p	insert the port number of the virtual host
%M.N	insert (part of) the name

M and N may have positive or negative integer values, which have the following meanings (see Table 4-2).

Table 4-2. *Meanings of variable values*

Value	Meaning
0	The whole name
1	The first part of the name
-1	The last part of the name
2	The second part of the name
-2	The next-to-last part of the name
2+	The second, and all following, parts
-2+	The next-to-last, and all preceding, parts

When the value is placed in the first part of the argument—in the M part of %M.N—it refers to parts of the hostname itself. When used in the second part—the N—refers to a particular letter from that part of the hostname. For example, in hostname `www.example.com`, the meanings of the variables are as shown in Table 4-3.

Table 4-3. *Example values for the hostname `www.example.com`*

Value	Meaning
%0	<code>www.example.com</code>
%1	<code>www</code>
%2	<code>example</code>
%3	<code>com</code>
%-1	<code>com</code>
%-2	<code>example</code>
%-3	<code>www</code>
%-2.1	<code>e</code>
%-2.2	<code>x</code>
%-2.3+	<code>ample</code>

Depending on the number of virtual hosts, you may wish to create a directory structure subdivided alphabetically by domain name, by top-level domain, or simply by host-name.

Note that *mod_vhost_alias* does not set the *DOCUMENT_ROOT* environment variable, and so applications that rely on this value may not work in this kind of virtual hosting environment.

See Also

- http://httpd.apache.org/docs/2.2/mod/mod_vhost_alias.html
- <http://httpd.apache.org/docs/2.2/vhosts/>

4.7 Mass Virtual Hosting Using Rewrite Rules

Problem

Although there is a module—*mod_vhost_alias*—which is explicitly for the purpose of supporting large numbers of virtual hosts, it is very limiting and requires that every virtual host be configured exactly the same way. You want to support a large number of vhosts, configured dynamically, but, at the same time, you want to avoid *mod_vhost_alias*.

Solution

Use directives from *mod_rewrite* to map to a directory based on the hostname:

```
RewriteEngine on
RewriteCond    "%{HTTP_HOST}"    "^ (www\.)? ([^\.]+)\.com"
RewriteRule    "%{.}*"           "/home/%2$1"
```

Discussion

mod_vhost_alias is useful, but it is best for settings where each virtual host is identical in every way but the hostname and document tree. Using *mod_vhost_alias* precludes the use of other URL-mapping modules, such as *mod_userdir*, *mod_rewrite*, and *mod_alias*, and it can be very restrictive. Using *mod_rewrite* is less efficient, but it is more flexible.

For example, when using *mod_vhost_alias*, you must do all of your hosts with *mod_vhost_alias*; whereas with this alternate approach, you can do some of your hosts using the rewrite rules and others using conventional virtual host configuration techniques.

The directives in the Solution map requests for *www.something.com* (or without the *www*) to the directory */home/something*.

See Also

- Recipe 5.17
- <http://httpd.apache.org/docs/2.2/vhosts/>
- http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

4.8 Logging for Each Virtual Host

Problem

You want each virtual host to have its own logfiles.

Solution

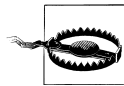
Specify *Errorlog* and *CustomLog* within each virtual host declaration:

```
<VirtualHost *:80>
    ServerName    waldo.example.com
    DocumentRoot  /home/waldo/www/htdocs

    ErrorLog      /home/waldo/www/logs/error_log
    CustomLog     /home/waldo/www/logs/access_log combined
</VirtualHost>
```

Discussion

The various logging directives can be placed either in the main body of your configuration file or within a *<VirtualHost>* section. When they are placed within a virtual host, log entries for that virtual host go in the specified logfiles, rather than into the log file(s) defined in the main server configuration.



Each logfile counts against the total number of files and network connections your server is allowed to have. If you have 100 virtual hosts, each with its own error and activity log, that's 200 open channels—and if the server's quota is 256, you can only handle 56 concurrent requests at any one time.

Those numbers are just examples; actual values for maximum open file quotas vary by platform, but are generally *much* larger. Consult your platform's documentation to find out your actual limit.

For this reason, we recommend that you have all your virtual hosts log to the same files, and split them apart later for analysis or examination.

In the recipe given here, the logfiles are placed within the home directory of a particular user, rather than in the main log directory. This gives you easier access to those files, but you still need to take adequate precautions to set the permissions on the directory in question. Consult Chapter 6 for a discussion on file permissions.

See Also

- Chapter 3
- Chapter 6
- Recipe 4.9

4.9 Splitting Up a LogFile

Problem

Because of a large number of virtual hosts, you want to have a single logfile for all of them and split it up afterward.

Solution

```
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost  
CustomLog logs/vhost_log vhost
```

Then, after rotating your logfile:

```
split-logfile < logs/vhost_log
```

Discussion

The *LogFormat* directive in this recipe creates a logfile that is similar to the common log file format but additionally contains the name of the virtual host being accessed. The *split-logfile* utility splits up this logfile into its constituent virtual hosts.

See Also

- Recipe 3.11

4.10 Port-Based Virtual Hosts

Problem

You want to present different content for HTTP connections on different ports.

Solution

Explicitly list the port number in the *<VirtualHost>* declaration:

```
Listen 8080  
  
<VirtualHost 10.0.1.2:8080>  
    DocumentRoot /www/vhosts/port8080  
</VirtualHost>
```

```
Listen 9090
```

```
<VirtualHost 10.0.1.2:9090>  
    DocumentRoot /www/vhosts/port9090  
</VirtualHost>
```

Discussion

Port-based virtual hosting is somewhat less common than other techniques shown in this chapter. However, there are a variety of situations in which it can be useful. If you have only one IP address, have no ability to add hostnames to DNS, or if your ISP blocks inbound traffic on port 80, it may be useful to run virtual hosts on other ports.

It also may be useful in development environments to run separate httpd instances on different ports for different developers or different setups.

Visitors to your Web site must list the port number in the URL that they use. For example, to load content from the second virtual host previously listed, the following URL might be used:

```
http://server.example.com:9090/
```

See Also

- <http://httpd.apache.org/docs/2.2/vhosts/>

4.11 Displaying the Same Content on Several Addresses

Problem

You want to have the same content displayed on two of your addresses.

Solution

Specify both addresses in the `<VirtualHost>` directive:

```
NameVirtualHost 192.168.1.1:80  
NameVirtualHost 172.20.30.40:80  
  
<VirtualHost 192.168.1.1:80 172.20.30.40:80>  
    DocumentRoot /www/vhosts/server  
    ServerName server.example.com  
    ServerAlias server  
</VirtualHost>
```

Discussion

This setup is most useful on a machine that has addresses that are internal to your network, as well as those that are accessible only from outside your network. If these are the only addresses, you could use the `*` notation introduced in Recipe 4.1. However,

if there are more addresses, this allows you to specify what content appears on what address.

See Also

- <http://httpd.apache.org/docs/2.2/vhosts/>

4.12 Defining virtual hosts in a database

Problem

You want to define virtual hosts in a database, rather than having to edit the configuration file each time

Solution

Obtain `mod_vhost_dbi` from http://www.outoforder.cc/projects/apache/mod_vhost_dbi/ and use that to configure virtual hosts to be loaded out of the database:

```
PoolDbiDriver      Server1  mysql
PoolDbiHost        Server1  192.168.1.50
PoolDbiUsername     Server1  datauser
PoolDbiPassword     Server1  password
PoolDbiDBName       Server1  vhosts
PoolDbiConnMin      Server1  1
PoolDbiConnSoftMax  Server1  1
PoolDbiConnHardMax  Server1  5
PoolDbiConnTTL      Server1  30
```

```
<VirtualHost *:80>
  VhostDbiEnabled On
  VhostDbiConnName Server1
  VhostDbiQuery "SELECT ServerName, DocumentRoot, Username " \
    FROM vhost_info WHERE ServerName = &{RequestHostname}"
</VirtualHost>
```

Discussion

`mod_vhost_dbi` is a third-party module that provides the ability to put your virtual host configuration in a database, and update those virtual hosts without having to modify your configuration file or restart Apache.

The full documentation, and code, for this module is available at http://www.outoforder.cc/projects/apache/mod_vhost_dbi/. Using the sample configuration above, you can get virtual hosts running. You'll need to create records in the `vhost_info` table for each virtual host, containing `ServerName`, `DocumentRoot` and `Username` where **`ServerName`** and `DocumentRoot` have the obvious meanings, and `Username` refers to the user id under which suexec will execute CGI processes.

Unfortunately, *mod_vhost_dbi* is a somewhat limited solution, because it provides only these three configuration directives.