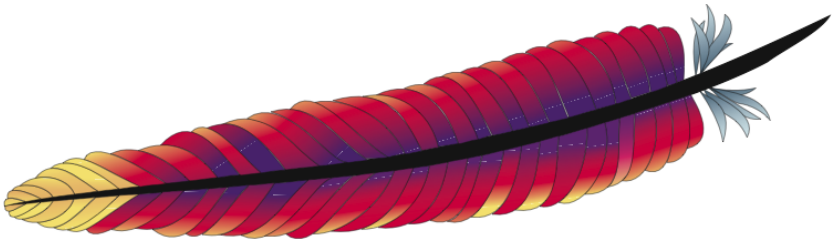
Apache HTTP Server



Tecnologie Internet

a.a. 2022/2023

History

In February of 1995, the most popular server software on the Web was the public domain HTTP daemon developed by Rob McCool at the National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana-Champaign.

However, development of that httpd had stalled after Rob left NCSA in mid-1994.

Using **NCSA httpd 1.3** as a base, a group of webmasters added all of the published bug fixes and worthwhile enhancements they could find, tested the result on their own servers, and made **the first official public release (0.6.2) of the Apache server** in April 1995.

History

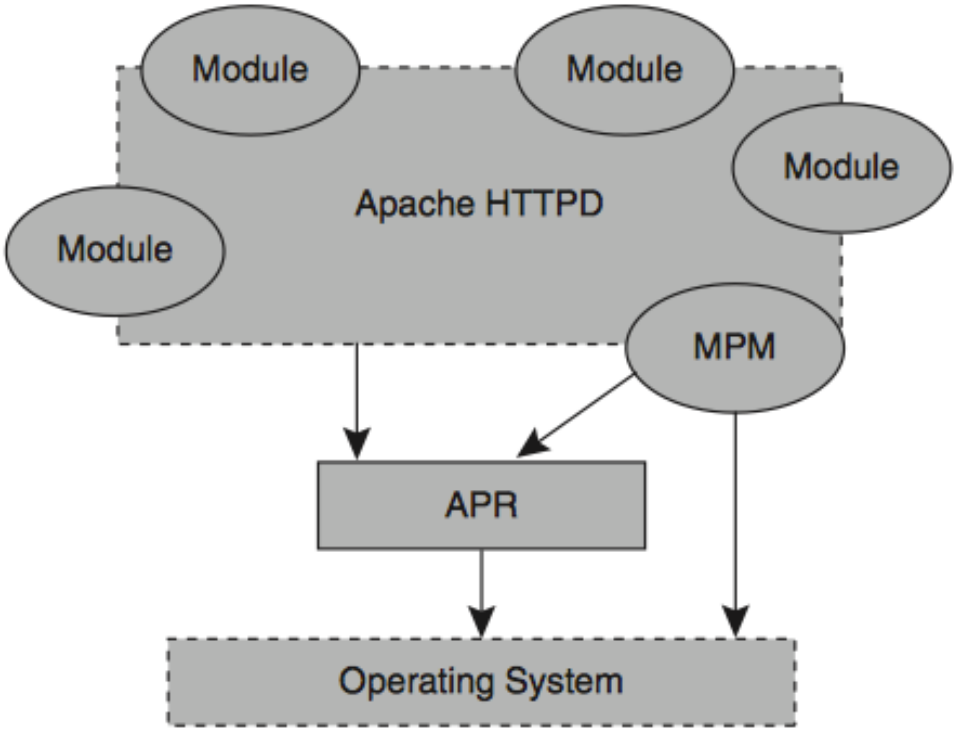
Apache 1.0 was released on December 1, 1995.

Less than a year after the group was formed, the Apache server passed NCSA's httpd as the #1 server on the Internet and according to the survey by Netcraft, it retains that position today (it is used on more than 50% active Internet sites).

Current version: **Apache 2.4**

In 1999, members of the Apache Group formed the **Apache Software Foundation** to provide organizational, legal, and financial support for the Apache HTTP Server. The foundation has placed the software on a solid footing for future development, and greatly expanded the number of Open Source software projects, which fall under this Foundation's umbrella.

Architecture



Modules

Modules may be compiled statically into the server or, more commonly, held in a /modules/ or /libexec/ directory and loaded dynamically at runtime.

Apache httpd

Implements the **request processing cycle**, including several **phases** (request parsing, security, preparation, handler).

Multi-Processing Module (MPM)

Intermediate layer between Apache and the underlying o.s., for managing processes (threads).

Apache Portable Runtime (APR)

Libraries that provide a cross-platform operating system layer and utilities, so that modules don't have to rely on non-portable operating system calls.

Multi-processing / multi-theading

Apache 2.x uses one process (or thread, depending on the configuration) per connection, with blocking I/O.

The server ships with a selection of **Multi-Processing Modules (MPMs)** which are responsible for binding to network ports on the machine, accepting requests, and dispatching children to handle the requests.

http://httpd.apache.org/docs/2.4/mpm.html

Benefits:
- Apache httpd can more cleanly and efficiently support a wide variety of operating systems. In particular, the Windows version of the server is now much more efficient, since mpm_winnt can use native networking features in place of the POSIX layer used in Apache httpd 1.3.
- The server can be better customized for the needs of the particular site. For example, sites that need a great deal of scalability can choose to use a threaded MPM like worker or event, while sites requiring stability or compatibility with older software can use a prefork.

Compiling and installing

Download
See http://httpd.apache.org/download.cgi
Extract
$ gzip -d httpd-*NN*.tar.gz
$ tar xvf httpd-*NN*.tar
$ cd httpd-*NN*
Configure
$ ./configure --prefix=*PREFIX*
Compile
$ make
Install
$ make install
Customize
$ vi *PREFIX*/conf/httpd.conf
Start
$ *PREFIX*/bin/apachectl -k start

http://httpd.apache.org/docs/2.4/install.html

NN must be replaced with the current version number, and *PREFIX* must be replaced with the filesystem path under which the server should be installed. If *PREFIX* is not specified, it defaults to /usr/local/apache2.

Configuration files and directives

The Apache HTTP Server is configured via **simple text files**. These files may be located any of a variety of places, depending on how exactly you installed the server.

The default location of the configuration files is **/usr/local/apache2/conf**. The default configuration file is usually called **httpd.conf**.

The configuration is frequently broken into multiple smaller files, for ease of management. These files are loaded via the **Include directive**.

The server is configured by placing **configuration directives** in these configuration files. A directive is a keyword followed by one or more arguments that set its value.

http://httpd.apache.org/docs/2.4/mod/directives.html

Configuration files and directives

Apache httpd 2.4 default layout:

```
ServerRoot                :: /usr/local/apache2
DocumentRoot              :: /usr/local/apache2/htdocs
Apache Config File        :: /usr/local/apache2/conf/httpd.conf
Other Config Files        :: /usr/local/apache2/conf/extra/

SSL Config File           :: /usr/local/apache2/conf/extra/httpd-ssl.conf

ErrorLog                  :: /usr/local/apache2/logs/error_log

AccessLog                 :: /usr/local/apache2/logs/access_log

cgi-bin                   :: /usr/local/apache2/cgi-bin
(enabled by default, but some of the bundled scripts are 644)

binaries (apachectl)      :: /usr/local/apache2/bin

start/stop                :: /usr/local/apache2/bin/apachectl
(start|restart|graceful|graceful-stop|stop|configtest)
```

Minimal configuration

```
User webuser

Group webgroup

ServerName machinename

DocumentRoot /usr/local/apache2/htdocs

Listen 80
```

User directive

The User directive sets the user ID as which the server will answer requests.

Syntax:
User *unix-userid*

Unix-userid is one of:

a username
Refers to the given user by name.

 # followed by a user number
Refers to a user by its number.

The user should have no privileges that result in it being able to access files that are not intended to be visible to the outside world, and similarly, the user should not be able to execute code that is not meant for HTTP requests.

Security
Don't set User to root unless you know exactly what you are doing, and what the dangers are.

Group directive

The Group directive sets the group under which the server will answer requests. In order to use this directive, the server must be run initially as root. If you start the server as a non-root user, it will fail to change to the specified group, and will instead continue to run as the group of the original user.

Syntax:
Group *unix-group*

Unix-group is one of:

A group name
Refers to the given group by name.

 # followed by a group number
Refers to a group by its number.

It is recommended to set up a new group specifically for running the server.

ServerName directive

The ServerName directive sets the request scheme, hostname and port that the server uses to identify itself.

Syntax:
ServerName [*scheme*://]*fully-qualified-domain-name*[:*port*]

When using **name-based virtual hosts**, the ServerName inside a **<VirtualHost>** section specifies what hostname must appear in the request's Host: header to match this virtual host.

Hostnames and DNS

In order to connect to a server, the client will first have to resolve the hostname to an IP address - the location on the Internet where the server resides. Thus, in order for your web server to be reachable, it is necessary that the hostname be in the **Domain Name System (DNS)**.
If you don't know how to do this, you'll need to contact your **network administrator**, or Internet service provider, to perform this step for you.

IP-based virtual hosts use the IP address of the connection to determine the correct virtual host to serve. Therefore you need to have a separate IP address for each host.

With **name-based virtual hosting**, the server relies on the client to report the hostname as part of the HTTP headers. Using this technique, many different hosts can share the same IP address.

Virtual hosts

IP-based Virtual Hosts (An IP address for each web site)

```
<VirtualHost 172.20.30.40:80>
    ServerAdmin webmaster@www1.example.com
    DocumentRoot "/www/vhosts/www1"
    ServerName www1.example.com
    ErrorLog "/www/logs/www1/error_log"
    CustomLog "/www/logs/www1/access_log" combined
</VirtualHost>

<VirtualHost 172.20.30.50:80>
    ServerAdmin webmaster@www2.example.org
    DocumentRoot "/www/vhosts/www2"
    ServerName www2.example.org
    ErrorLog "/www/logs/www2/error_log"
    CustomLog "/www/logs/www2/access_log" combined
</VirtualHost>
```

Virtual hosts

Name-based Virtual Hosts (More than one web site per IP address)

```
<VirtualHost *:80>
    # This first-listed virtual host is also the default for *:80
    ServerName www.example.com
    ServerAlias example.com
    DocumentRoot "/www/domain"
</VirtualHost>

<VirtualHost *:80>
    ServerName other.example.com
    DocumentRoot "/www/otherdomain"
</VirtualHost>
```

Hostnames and DNS

If you are testing a server that is not Internet-accessible, you can put host names in your **hosts file** in order to do local resolution. For example, you might want to put a record in your hosts file to map a request for www.example.com to your local system, for testing purposes. This entry would look like:

127.0.0.1 www.example.com

A hosts file will probably be located at /etc/hosts or C:\Windows\system32\drivers\etc\hosts.

DocumentRoot directive

This directive sets the directory from which Apache httpd will serve files. Unless matched by a directive like **Alias**, the server appends the path from the requested URL to the document root to make the path to the document.

Syntax:
DocumentRoot *directory-path*

Example:

DocumentRoot "/usr/web"

then an access to http://my.example.com/index.html refers to /usr/web/index.html. If the *directory-path* is not absolute then it is assumed to be relative to the **ServerRoot directive**, which sets the directory in which the server lives.

Listen directive

The Listen directive instructs Apache httpd to accept incoming requests only on the specified port(s) or address-and-port combinations. Listen is now a required directive. If it is not in the config file, the server will fail to start. This is a change from previous versions of Apache httpd.

Syntax:
Listen [*IP-address*:]*portnumber* [*protocol*]

Multiple Listen directives may be used to specify a number of addresses and ports to listen to. For example, to make the server accept connections on both port 80 and port 8000, on all interfaces, use:
Listen 80
Listen 8000

To make the server accept connections on port 80 for one interface, and port 8000 on another, use:
Listen 192.170.2.1:80
Listen 192.170.2.5:8000

IPv6 addresses must be surrounded in square brackets, as in the following example:

Listen [2001:0db8:85a3:0000:1319:8a2e:0370:7344]:80

ErrorLog directive

For an Apache HTTP Server administrator, the most valuable assets are the log files, and, in particular, the error log.

The location of the error log is defined by the **ErrorLog directive**, which may be set globally, or per virtual host. Entries in the error log tell the admin what went wrong, and when. They often also tell the admin how to fix it.

Syntax:
ErrorLog *file-path*|syslog[:*facility*]

Each error log message contains an error code. The admin can also configure his/her error log to contain a log ID which he/she can then correlate to an access log entry, to determine what request caused the error condition.

CustomLog directive

The CustomLog directive is used to log requests to the server. A log format is specified, and the logging can optionally be made conditional on request characteristics using environment variables.

Syntax:
CustomLog *file*|*pipe format*|*nickname* [env=[!]*environment-variable*| expr=*expression*]

The first argument, which specifies the location to which the logs will be written, can take one of the following two types of values:

file
A filename, relative to the ServerRoot.

pipe
The pipe character "|", followed by the path to a program to receive the log information on its standard input.

The second argument specifies what will be written to the log file. It can specify either a *nickname* defined by a previous **LogFormat directive**, or it can be an explicit *format* string.

The third argument is optional and controls whether or not to log a particular request.

Include directive

This directive allows inclusion of other configuration files from within the server configuration files.

Syntax:
Include *file-path*|*directory-path*|*wildcard*

Shell-style (fnmatch()) wildcard characters can be used in the filename or directory parts of the path to include several files at once, in alphabetical order. However, including entire directories is not recommended, because it is easy to accidentally leave temporary files in a directory that can cause httpd to fail. Instead, use the wildcard syntax shown below.

The Include directive will **fail with an error** if a wildcard expression does not match any file. The **IncludeOptional directive** can be used if non-matching wildcards should be ignored.

The file path specified may be an absolute path, or may be relative to the **ServerRoot** directory.

```
Include /usr/local/apache2/conf/ssl.conf
Include /usr/local/apache2/conf/vhosts/*.conf
Include conf/vhosts/*/*.conf
IncludeOptional conf/vhosts/*/*.conf
```

LoadModule directive

The LoadModule directive links in the object file or library *filename* and adds the module structure named *module* to the list of active modules.

Syntax:
LoadModule *module filename*

module is the name of the external variable of type module in the file, and is listed as the **Module Identifier** in the module documentation.

Example:

LoadModule status_module modules/mod_status.so

loads the named module from the modules subdirectory of the ServerRoot.

https://modules.apache.org

IfModule directive

The <IfModule *test*>...</IfModule> section is used to mark directives that are conditional on the presence of a specific module. The directives within an <IfModule> section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

Syntax:
<IfModule [!]*module-file*|*module-identifier*> ... </IfModule>

The *test* in the <IfModule> section directive can be one of two forms:

- *module*
- !*module*

In the former case, the directives between the start and end markers are only processed if the module named *module* is included in Apache httpd -- either compiled in or dynamically loaded using **LoadModule**.

The second format reverses the test, and only processes the directives if *module* is **not** included.

“Local” configurations

To apply directives to specific parts of the server:

<Directory></Directory>

<File></File>

<VirtualHost XYZ></VirtualHost>

<FilesMatch REGEX></FilesMatch>

```
<Directory /var/www/html>
     Options Indexes
     AllowOverride none
     Order deny,allow
     deny from all
     allow from example.org localhost
</Directory>
<FilesMatch ^\.ht>
   Order deny,allow
   deny from all
</FilesMatch>
```

Options directive

The Options directive controls which server features are available in a particular directory.

Syntax:
Options [+|-]*option* [[+|-]*option*] ...

option can be set to None, in which case none of the extra features are enabled, or one or more of the following:

All
All options except for MultiViews.

ExecCGI
Execution of CGI scripts using **mod_cgi** is permitted.

FollowSymLinks
The server will follow symbolic links in this directory. This is the default setting.

SymLinksIfOwnerMatch
The server will only follow symbolic links for which the target file or directory is owned by the same user id as the link.

Options directive

…

Includes
Server-side includes provided by **mod_include** are permitted.

IncludesNOEXEC
Server-side includes are permitted, but the #exec cmd and #exec cgi are disabled. It is still possible to #include virtual CGI scripts from **ScriptAlias**ed directories.

Indexes
If a URL which maps to a directory is requested and there is no **DirectoryIndex** (*e.g.*, index.html) in that directory, then **mod_autoindex** will return a formatted listing of the directory.

.htaccess files

.htaccess files **provide a way to make configuration changes on a per-directory basis**.

A file, containing one or more configuration directives, is placed in a particular document directory, and the directives apply to that directory, and all subdirectories thereof.

It is better to avoid using .htaccess files completely, having access to httpd main server config file. Using .htaccess files slows down the Apache http server. Any directive that can be included in a .htaccess file is better set in a **Directory** block, as it will have the same effect with better performance.

Performance tuning

Apache 2.x is a general-purpose web server, designed to provide a balance of flexibility, portability, and performance. Although it has not been designed specifically to set benchmark records, Apache 2.x is capable of high performance in many real-world situations.

There are some options that a server administrator can configure to tune the performance of an Apache 2.x installation.

The single biggest hardware issue affecting webserver performance is **RAM**. A webserver should never ever have to swap, as swapping increases the latency of each request beyond a point that users consider "fast enough".

For run-time and compile-time configuration issues, see:

http://httpd.apache.org/docs/2.4/misc/perf-tuning.html

ErrorDocument directive

In the event of a problem or error, Apache httpd can be configured to do one of four things,

1) output a simple hardcoded error message
2) output a customized message
3) internally redirect to a local *URL-path* to handle the problem/error
4) redirect to an external *URL* to handle the problem/error

The first option is the default, while options 2-4 are configured using the ErrorDocument directive, which is followed by the HTTP response code and a URL or a message. Apache httpd will sometimes offer additional information regarding the problem/error.

Syntax:
ErrorDocument *error-code document*

```
ErrorDocument 500 http://foo.example.com/cgi-bin/tester
ErrorDocument 404 /cgi-bin/bad_urls.pl
ErrorDocument 401 /subscription_info.html
ErrorDocument 403 "Sorry can't allow you access today"
ErrorDocument 403 Forbidden!
ErrorDocument 403 /cgi-bin/forbidden.pl?referrer=%{escape:%{HTTP_REFERER}}
```

Starting the server

On Unix, the httpd program is run as a **daemon** that executes continuously in the background to handle requests.

If the **Listen directive** specified in the configuration file is default of 80 (or any other port below 1024), then it is necessary to have root privileges in order to start apache, so that it can bind to this privileged port.

Once the server has started and performed a few preliminary activities such as opening its log files, it will launch several ***child processes*** which do the work of listening for and answering requests from clients. The main httpd process continues to run as the root user, but the child processes run as a less privileged user.

Starting the server

The **apachectl** control script sets certain environment variables that are necessary for httpd to function correctly under some operating systems, and then invokes the httpd binary.

The first thing that httpd does when it is invoked is to locate and read the configuration file **httpd.conf**.

If all goes well during startup, the server will detach from the terminal and the command prompt will return almost immediately. This indicates that the server is up and running.

It is then possible to use the browser to connect to the server and view the **test page** in the DocumentRoot directory.

Stopping and restarting

1) use kill to send a signal to the main process of the server:

```
kill -TERM `cat /usr/local/apache2/logs/httpd.pid`
```

2) use apachectl -k with one of the following options: stop, restart, graceful and graceful-stop

References

http://httpd.apache.org/docs/2.4/

http://httpd.apache.org/docs/2.4/mpm.html

http://httpd.apache.org/docs/2.4/mod/directives.html

http://httpd.apache.org/docs/2.4/misc/perf-tuning.html