



# Getting Up and Running with Apache 2.0 on Linux

by Paul Weinstein  
Chief Consultant  
Waubonsie Consulting  
A Hands-On Lab for  
LinuxWorld New York City  
January 20th, 2004

Welcome, to the Apache 2.0 on Linux Hands-on Lab. I'll talk a little about how this lab will work and what we'll be covering in a few moments, but first, may I take this moment to wish you well and welcome to winter in New York City.

The only thing worst than doing LinuxWorld in NYC in January is doing LinuxWorld in San Francisco in August. As Mark Twain once quipped, "The coldest winter I ever spent was a summer in San Francisco."



# Introduction

“i am who i am who i am who am i  
requesting some enlightenment  
could I have been anyone other than  
me?”

-from Dancing Nancies, Dave Matthews Band





# Introduction

- the topic at hand: apache 2.0
  - what we'll cover in the next 3 hours
    - history of
    - getting our hands on
    - what's on hand
    - configuring, building, migrating to
    - adding third-party modules
    - fine tuning
    - future of





# Introduction

- the topic at hand: apache 2.0
  - what we won't cover
    - detailed changes in apache source code, apache APIs
    - workings of apache modules
    - security precautions
    - apache in relation to a specific linux distribution





# Introduction

- the topic at hand: apache 2.0
  - how this hands-on lab will work
    - build, configure latest apache 2.0 release on our linux laptops
    - 2, 15 minute breaks
    - workbook, which contains a copy of the slides you see and additional information such as notes, comments and resources that further illuminate the topic at hand.
    - with 3 hours to spare, feel free to stop and ask questions.





# Apache 2.0: A History

“history is the version of past events that people have decided to agree upon”

-Napoleon Bonaparte

The following summary of Apache 2.0's history is from the invaluable Apache resource Apache Week. Apache Week's website is <<http://www.apacheweek.com>> and all things Apache 2.0 can be found at: <http://www.apacheweek.com/features/ap2>



# Apache 2.0: A History

- plans for apache 2.0 date as far back as 1996

Of course when one considers that fundamentally Apache 2.0 is a total rewrite of the Apache Web Server we start to understand why Apache 2.0 is a little long in the tooth. Why commit to a total code rewrite considering how successful the Apache Web Server has been over the last few years?

Consider that the Apache Web Server got its name from being an “A-Patchy” Web Server, or better put a set of patches for adding functionality to what was, at the time, the leading web server of choice, the NCSA Web Server. While the code itself may no longer resemble the NCSA Web Server, the Apache Web Server code is still, fundamentally a set of patched code.

This state of patch work code of course presents many difficulties in terms of adding new functionality, as will discuss in a moment, and maintenance. With these two issues in mind, we can understand why discussions for what we now call Apache 2.0 took place as early as 1996.



# Apache 2.0: A History

- in june 1997 an official set of requirements for rewriting the core apache code was put forth
- some of the issues discussed at the time included:
  - portability
  - scalability
  - configuration
  - I/O filtering

So what requirements did the Apache Group set for their code redesign? High on their list was dealing with issues of code portability and server scalability. This may strike us as surprising considering the Apache Web Server's legendary abilities to scale from serving personalized static web pages to running large dynamically driven Enterprise-level websites. Nonetheless, Apache is known to breakdown on the very-high end scale when running on certain platforms.

This of course, in turn, brings up the issue of portability. While Apache has been successful at being ported to numerous platforms, this has come at a high price, with each new platform to support the core code has seen a new set of `ifdef` statements, adding to the difficulty of maintenance and development. Moreover, while Apache's prefork processing model works well for Unix-type systems such as Linux, it cannot be implemented on all platforms, such as Windows - bring us back to the issue of scalability and resilience.

Configuration has also been adversely affected by the patchwork code. Many directives within the Apache configuration file are redundant, operating on options that other directives also operate on. While this redundancy can be seen as a positive, in most cases it has simply lead to confusion in regards to which directive to use and why.

Lastly, the issue of I/O processing needed to be rectified. Traditionally, Apache modules have written their output directly to the TCP connection. This solution however, lacks flexibility. Consider the problem of having to add an SSL/TLS layer for encryption. On the surface this may not seem to be such a major problem. The core Apache code handles the bulk of the transaction, an SSL aware module adds on the necessary layer and off everything goes to the client. But what if we added PHP? Now the SSL module must intercept all traffic between the client and the PHP module. Add another module, more problems.



# Apache 2.0: A History

- in june 1998 the apache core developers met for the first time to discuss the organizational structure of the apache group as well

As with any thoughtful code review, the Apache Group considered the organizational structure that should be use for the Apache Web Server and related projects.

To that end the formation of the Apache Software Foundation, a legal non-for-profit organization, was created to ensure, as the Apache website currently states, “that the Apache projects continue to exist beyond the participation of individual volunteers, to enable contributions of intellectual property and funds on a sound basis, and to provide a vehicle for limiting legal exposure while participating in open-source software projects.”

The Apache Software Foundation is made up of a group of Apache Software Foundation members, “Individuals who have demonstrated a commitment to collaborative open-source software development, through sustained participation and contributions within the Foundation's projects” who are “awarded membership after nomination and approval by a majority of the existing ASF members. Thus, the ASF is governed by the community it most directly serves -- the people collaborating within its projects.”

Moreover, the Apache Software Foundation maintains, per its non-for-profit charter, a “Board of Directors to manage the organizational affairs of the Foundation, as accorded by the ASF Bylaws. The Board, in turn, appoints a number of officers to oversee the day-to-day operations of the Foundation.”

Additional information about the Apache Software Foundation can be found at:  
<<http://www.apache.org/foundation/>>



# Apache 2.0: A History

- in january 2000, 2.0 became the primary development branch for the apache web server project

The first milestone in the development of the new Apache 2.0 code base came with the creation of the Apache 2.0 development tree in January 2000.

The decision to create a development branch for Apache 2.0 came from the Apache Group discussion about how to deal with feature additions to the stable Apache 1.3. Specifically, that all attention should be placed on 2.0 development and that no major new features would be accepted into the Apache 1.3 tree.



# Apache 2.0: A History

- the first apache 2.0 alpha was launched at the apachecon 2000 conference in march 2000

The second milestone for Apache 2.0 was the first Alpha release at ApacheCon 2000. There, a number of ASF members, on stage, updated the website, live of in front of the audience.



# Apache 2.0: A History

- the first apache 2.0 beta was launched at the apachecon 2001 conference in march 2001

Yet another milestone was reached a year later when, again at ApacheCon, the first beta release of Apache 2.0 was made live. This first beta was followed by a second Apache 2.0 beta in mid-November 2001.

In between the two beta releases a large amount of internal code changes occurred along with a few alpha-quality releases.



# Apache 2.0: A History

- apache 2.0 “went gold” with the first general availability release of version 2.0.35 in april 2002

Following a third beta released in February 2002, the first “general availability release” of Apache 2.0, dubbed Apache 2.0.35, went out onto the world in April.



# Apache 2.0: A History

- in november 2002, it was decided that development of new features would take place in a 2.1 development tree

To complete the cycle, the discussion within the Apache group in November of 2002 focused on how to deal with feature additions to the stable Apache 2.0 code. Again, it was decided that all attention should be placed on a development tree, this time dubbed Apache 2.1, and that no major new features would be accepted into the Apache 2.0 tree.

As such, the Apache 2.0 has become the stable release of the Apache Web Server with Apache 2.1 as the main development branch. Apache 1.3 is simply in maintenance mode, probably until Apache 2.0 has reach a critical mass in its adoption rate.

Since the tagging of Apache 2.0 as stable a number of commit restrictions have been set in motion to insure its stability. A new policy requires that all patches first be submitted to the mailing list for review before being committed. If the changes receive positive feedback from three committers or the suggested changes receive no negative feedback within a determined set of days, the submitted patches can be committed.

All of this can be seen to the general world in one of the most important areas of Apache 2.0, the API. The API for Apache 2.0, which was still undergoing changes after the first “general availability release”, has been kept stable since the release of version 2.0.42



# Apache 2.0 History

- as of 17th of nov 2003 the latest stable release of apache 2.0.48
  - apache 2.0.48 was released on 29th of oct 2003

A complete listing of all Apache 2.0 releases, along with comments describing the nature of the release, can be found at Apache Week, <http://www.apacheweek.com/features/ap2#rh>



# New and Improved

“but wait, there's more!”

-from any given infomercial

The following summary of Apache 2.0 enhancements comes from:

- \* Linux Journal Summary of Apache 2.0,  
<<http://www.linuxjournal.com/modules.php?op=modload&name=NS-articles/misc/images&file=4559s1?>>

- \* From Ryan Bloom's ApacheCon Europe 2000 presentation "Apache 2.0",  
<<http://www.rkbloom.net/presentation/Apache-2.0/img0.htm>>

- \* Rich Bowen's O'Reilly Open Source Convention 2002 Tutorial "Migrating to Apache 2.0", <<http://www.apache.org/~rbowen/presentations/apache20migration.pdf>>

A formal overview of Apache 2.0 enhancements can be found at:

<[http://httpd.apache.org/docs-2.0/new\\_features\\_2\\_0.html](http://httpd.apache.org/docs-2.0/new_features_2_0.html)>



# New and Improved

- new process support with multiple-processing modules (MPMs)
  - options vary depending on platform
  - on linux
    - prefork
    - prefork with threads (worker)
    - perchild

Now that we know the goals the Apache Group set out to reach, the question becomes how did they meet them?

In regards to a new processing model the Apache Group determined that embracing the Apache Web Server's modular design was the best solution and thus the Multiple Processing Modules was born. MPMs, besides meeting the stated goal of implementing a better solution for handling multiple process models, also helps Apache 2.0 in reaching a greater OS independence in the core code, a second goal for Apache 2.0 we'll discuss in a moment.

Currently, the Apache Group supplies seven options for MPMs. Which of these seven processing modules to use will vary depending on which platform our Apache 2.0 Web Server is running on. For Linux we can concern ourselves with three specific processing types; Prefork, Prefork with Threading (also called 'Worker') and Perchild.

The Prefork method is nothing new to us as this is the method that the Apache Web Server has always used on Linux. On startup the Apache parent process, running as root, creates a number of child processes, the number of which and user type is predefined in the configuration file. These child processes handle request for our web server. If a spike of requests is beyond the allotment of currently running child process, the parent process will fork off more process to catch up. The root parent process is, however, limited to a predefined maximum number of child process since, among other things, creating additional child processes is a resource expensive and time consuming exercise. Prefork is the default MPM for Apache 2.0 on Linux.

The Prefork with Threads module, also know as 'Worker' is designed to be a hybrid multi-process, multi-threaded model offering "the best of worlds". As with Prefork, Worker will launch additional child processes to deal with increased server load. The catch is that Worker also allows for threading, thus we have several processes that have several threads running in parallel to each other. This multi-thread, multi-process model allows for greater scalability when needed. The downside is that Worker is slightly less robust as compared to Prefork, since a crashing process, thanks to the threaded nature, will take out all the currently established connections to the ill process. (The plus side of course is that the other child processes are unaffected, thus not all of the connections are lost, something that would happen in a traditional multithreaded server application).

Finally, we have the Perchild module. With this module there are a fix number of processes at startup. Each process can have a changing number of threads starting with a predefined number at runtime. That is, each child process has its own set of threads, spawning and reaping as needed based on server load, all of which is contained within the predefined confines of the configuration file.

A benefit of Perchild is the ability, using an Apache configuration option, to bind a particular process to a particular virtual host. Moreover, the server process can be bound to a specific user and group id. Allowing for a greater control over the assignment of resources based on need for specific virtual hosts.



# New and Improved

- the Apache Portable Runtime (APR) was introduced as an abstraction layer to masks the call differences of various systems
- the modules' APIs has introduced several enhancements to provide more flexibility in handling callback routines and data structures

This issue of portability, or more specifically the issue of portability as it relates to the core web server code, has been dealt with the introduction of an Apache Portable Runtime layer. The APR masks the differences of various supported platforms, allowing for a greater flexibility and control in porting. Moreover, support for non-UNIX platforms has been improved, and Apache is now faster and more stable on a larger number of platforms, thanks in part to the introduction of the platform-specific MPMs we just covered.

Of course Apache's APIs have changed significantly for 2.0 in order to implement various enhancements. Two of the major new features for Apache modules in 2.0 include ordering modules on a per-function basis instead of per-module one and the ability of modules to register functions that other modules can call. We'll discuss modules in relation to Apache 2.0 a bit later. Of important note for the moment is that the new API means Apache 1.3.x modules won't work with 2.0 without modification.



# New and Improved

- I/O filtering
  - one module can modify the results of another
  - filters can be configured within the config file
  - modules can add filters based on request
- the configuration file has changed
  - read once and stored in memory
  - easy to convert to XML
  - divided into multiple files

Another enhancement to the Apache API is that of I/O filtering. Filters allow for the ability to chain actions for processing data coming in or going out.

For example, with Apache 1.3 we can invoke a CGI from a Server Side Include token, but the token needs to reside within a static document that our Apache server knows to process with `mod_include`. We cannot, within the 1.3 model, send a SSI token from a CGI script to invoke the SSI-based action. This has to do with the fact that all modules, before Apache 2.0, write directly to the TCP stream. Thus we only get “one shot” at generating dynamic content - an all or nothing approach that does not provide an administrator with great flexibility. However, with Apache 2.0 we can invoke `mod_include` to process SSI tokens embedded in dynamic content.

The Apache configuration file, `httpd.conf` has been simplified to a degree by the house cleaning of dead, useless directives and of confusing overlapping functionality. Moreover, the configuration file has been divided into multiple files and has been designed for easy conversion to XML for greater configuration management options.

Previous to Apache 1.3, functions that needed to refer to options defined in the configuration file would read and process the configuration file as stored on the hard drive. This, despite the fact that the stored configuration file may not be the exact configuration that was actually used to load the running Apache processes. With Apache 2.0 the web server loads the configuration used at the invocation of the server into primary memory system. Thus anytime a module function needs to refer to a configuration option, the module access the data structure created by Apache, instead of the configuration file stored on the secondary memory system. The benefit is two-fold; The Apache server and modules will never be out-of-sync during run-time and the Apache server is less dependent on file access latency.

It should be noted that this change does not reflect a change in the processing of `htaccess` files. `.htaccess` files are still loaded each time a directory is traversed by the server on its way to serving a file residing in the same directory structure.



# New and Improved

- apache 2.0 also supports the new internet protocol, IPv6
- the apache group's distribution of apache 2.0 now includes additional modules such as mod\_ssl, mod\_dav, mod\_cgid

Some additional changes of note include the fact that Apache now supports Internet protocol, IPv6. On systems where IPv6 is supported by the underlying Apache Portable Runtime library, Apache gets IPv6 listening sockets by default. Moreover, the Listen, NameVirtualHost and <VirtualHost> directives support IPv6 numeric address strings (e.g., Listen [fe80::1]:80).

mod\_ssl, based off of Ralf S. Engelschall's mod\_ssl for Apache 1.3.x but now maintained by the Apache Group, is distributed within the Apache distribution thanks to the expiration of RSA patent and newer Export laws for open source software developed in the United States

Support for the WebDAV protocol, via mod\_dav is include with Apache 2.0

mod\_cgi still works for Apache 2.0, but is not optimal with threaded MPMs, hence mod\_cgid, which creates a simple Daemon process that, controls new CGI processes.



# Getting Apache 2.0

“it's rather like going to buy coke, complaining to the coca-cola company that the drink was too sour, then finding that your supermarket was adding lemon”

-“Vendor patches to Apache” Apache Week, 3rd March 2003

The following section covers the various sources for the Apache 2.0 Web Server and some of the problems and solutions that each source offers in relation to Apache on Linux.

For an introduction and perspective on this issue the following articles on Apache Week, “Vendor Patches to Apache” <<http://www.apacheweek.com/features/vendorversions>>, and NewFactor’s “Patching Apache”, <<http://www.newsfactor.com/perl/story/21560.html>>, maybe of interest.



# Getting Apache 2.0

- from linux distribution
  - Gentoo
  - Debian
  - Red Hat
  - S.u.S.E.

A number of Linux vendors have started to distribute, or at least make available, Apache 2.0. Some of these distributions include; Red Hat, SuSE, Debian and Gentoo.

The benefit of using a Linux vendor is that of having one source - your Linux vendor - for managing your platform and its packaged software. Moreover, many distributions optimize Apache for their specific distribution. For example, according to the Apache Week article "Vendor Patched for Apache," some of these customizations include "additional types using AddType directives in httpd.conf [and] custom init scripts." Apache Week also notes that of the Linux vendors they took note of, "Connectiva, Debian, EnGarde, Gentoo, Mandrake, Red Hat, and SCO all included a [security-related] patch for CAN-2001-0131, a vulnerability in htpasswd and htdigest that could allow local users to overwrite arbitrary files via a symlink attack. This vulnerability is not yet fixed in Apache, as it's tricky to get right cross-platform. The vendors patching this themselves only have to worry about the Linux architecture so can add a specific fix."

A few problems, however, do popup with using this method. First, we become depended on what the vendor has provided. Also, because Linux distributions sometime opt to backport security patches instead of providing the latest release from the Apache Group, its not always obvious on the first pass if an Apache Server is still vulnerable to a security bug or not. (Of course the obvious solution to this issue of backporing is to understand the vendors policy for security related patches. For example Red Hat has outlined its policy in relation to the Apache Web Server on its website, <[http://www.redhat.com/advice/speaks\\_backport.html](http://www.redhat.com/advice/speaks_backport.html)>). Moreover, it has not been unknown for a Linux vendor to create a security vulnerability of its own in customizing Apache. Therefore it should not be concluded that a Linux vendor's packaging of Apache is any more less secure compared to a Apache binary built from the source code provided at Apache Group's website.



# Getting Apache 2.0

- from `apache.org`
  - `http://httpd.apache.org/download.cgi`

The obvious advantage one has with open source software is the ability to get familiar with the source code and build process of specific piece of software.

Information about how the Apache Group recommends you verify the source code you downloaded can be found on the download page as well as at: <http://httpd.apache.org/dev/verification.html>. Because online resources can be cracked, it is always recommend that one verifies the information provided from multiple sources, preferably one of which is a trusted, offline source.



# Getting Apache 2.0

“the last thing anyone in the apache community seems to want is to compromise apache's open-source roots and cause fragmentation of a web server that is extremely popular. at the same time, most distributors want to make money on the software and fine-tune it to run with their linux distributions. walking that fine line is no easy task.”

-“Patching Apache” NewsFactor Network, 21st May 2003

So why is this even an issue? And why bring it up in a talk about Apache 2.0? Because this is an issue that is of particular relevance to users of Linux. To build your own custom binary from scratch or to use a vendor's customized build.

Moreover, as with distributing the Linux kernel, Apache distributors walk a fine line between optimizing the code and breaking compatibility. While no one wishes to fork Apache it is important to understand what "is Apache" and what is just "based on" or "derived from" Apache. The Apache Software Foundation helps in the case since, giving a legal foothold for the Apache developers to stand on. As noted in NewsFactor, "the Apache Software Foundation prefers that vendors that do [modify Apache, to] .. clearly indicate it in the software package's nomenclature."



# Building Apache 2.0

- “truly, you have a dizzying intellect.”
- “wait 'til I get going!”

- exchange from “The Princess Bride”

In any case, the only way to partake in the full advantage of the Apache Web Server is know and understand how the Apache Web Server works. This means taking a look at the code and understanding the “standard” distribution of the server from the Apache Software Foundation.

So, let us take a look at building Apache 2.0 from scratch and note how the changes we are discussing have been implemented as laid out by the Apache Group’s requirements and specifications.

Off, to our laptops shall we?



# Building Apache 2.0

- getting our bearings
  - apache file layout
    - no independent src/ directory
    - docs, config files, sample cgi scripts and error documents reside in docs/ directory
    - modules, os related items modules/ and os/ directories

As one may notice, the distribution tree for Apache, has changed. (Of note, from here on out, unless specifically noted, whenever a reference to Apache is made, it is in reference to the standard distribution of Apache 2.0 from the Apache Group). For example there is no independent src/ directory. The source code has been divided into relevant directories. Modules and their code can be found in the modules/ directory. OS specific code can be found in the os/ directory.

Documentation, configuration files, sample cgi scripts and error documents can now be found in the docs/ directory



# Building Apache 2.0

## module-go-round:

### - gone

- mod\_auth\_db
- mod\_digest
- mod\_log\_agent
- mod\_log\_referer

### -new

- mod\_cache
- mod\_auth\_cgid
- mod\_charset\_lite
- mod\_dav
- mod\_deflate
- mod\_ext\_filter
- mod\_file\_cache
- mod\_isapi
- mod\_ssl
- mod\_suexec

A number of modules have been removed, added or rewritten. Some modules of note are listed above, such as `mod_log_agent`, `mod_log_referer` and `mod_auth_db`. In the case of these modules the removal is designed to remove confusion in relation to configuration options that have superseded the need for these modules.

As previously noted some additions include `mod_ssl` and `mod_dav`



# Building Apache 2.0

`./configure` with no options:

- server path is: `/usr/local/apache2`

- modules as listed with `./httpd -l`

- `core.c`
- `mod_access.c`
- `mod_auth.c`
- `mod_include.c`
- `mod_log_config.c`
- `mod_env.c`
- `mod_setenvif.c`
- `prefork.c`
- `http_core.c`
- `mod_mime.c`
- `mod_status.c`
- `mod_autoindex.c`
- `mod_asis.c`
- `mod_cgi.c`
- `mod_negotiation.c`
- `mod_dir.c`
- `mod_imap.c`
- `mod_actions.c`
- `mod_userdir.c`
- `mod_alias.c`

By default the invocation of the Apache configure script will configure an Apache build with the modules listed above for the Linux platform. It will also install Apache in /usr/local/apache2.

Notice that the configuration script had Apache configured to use the Prefork MPM.



# Building Apache 2.0

- some additional options for `--configure`
  - `--enable-auth-digest`
  - `--enable-echo`
  - `--enable-mem-cache`
  - `--enable-case-filter`
  - `--enable-auth-ldap`
  - `--enable-usertrack`
  - `--enable-unique-id`
  - `--enable-proxy`
  - `--enable-info`
  - `--enable-vhost-alias`
  - `--enable-speling`
  - `--enable-rewrite`

The configuration script is by no means limited to just the modules previously shown. The Apache distribution includes many useful modules. The above list is by no means a complete list. This is however, to some extent, in addition to what has been covered in the last two slides.



# Building Apache 2.0

- dynamic or static modules  
    --enable-so                      DSO capability

As with previous versions of Apache, modules can be configured to load statically or dynamically. To configure Apache 2.0 to allow Dynamic Shared Objects, use the `--enable-so` switch when running the configure script.



# Building Apache 2.0

```
./configure --  
  prefix=/usr/local/apache-  
  2.0.48 --enable-so
```

```
make
```

```
sudo make install
```

If you wish to configure Apache for your Linux distribution's layout instead of installing Apache completely unto itself, you can use the `--enable-layout` switch. For example if you wish to use the standard GNU layout structure you can replace the `--prefix=` switch with `--enable-layout=GNU` or to enable the layout for Debian, `--enable-layout=Debian`. A complete list of the optional layouts can be found in the `config.layout` file.

Of final note in relation to the `configure` script, Apache 2.0 uses a more standard `autoconf` and `libtool` tools, allowing Apache's configuration system to behave in a similar manner to other source code packages.

Documentation on configuring and installing Apache can be found at:  
<<http://httpd.apache.org/docs-2.0/install.html>>



# Intermission

- 15 minutes while everyone's laptops run the apache build processes

So have at it ;-)



# Configuring Apache 2.0

“i am extraordinarily patient,  
provided I get my own way in  
the end”

- Margaret Thatcher

Detailed information about installing and running Apache 2.0 for various Linux distributions can be found online at the vendor's documentation site. For example, both Red Hat and SuSE provide information on their websites. They are <<http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-httpd.html>> and <<http://sdb.suse.de/en/sdb/html/apache2-faq.html>>, respectively.

Furthermore, when possible within this section a reference to the Apache documentation as it relates to the directive or configuration option as currently discussed has been added.

Please feel free to follow along in your freshly installed Apache 2.0 directory as we take a look at the latest version of Apache.



# Configuring Apache 2.0

- getting our bearings
  - apache file layout

bin/	modules/
cgi-bin/	build/
error/	conf/
icons/	htdocs/
lib/	include/
man/	logs/
	manual/

Nothing much in terms of the Apache directory layout once installed.

After a little looking around in these directories, let's have at the most important one, `conf/`.



# Configuring Apache 2.0

- global environment configuration
  - this section has seen a large number of changes compared with apache 1.3

As with Apache 1.3, Apache 2.0 breaks the configuration file into three main parts; A section for “Global Environment Configuration” - directives that affect the overall operation of our Apache Web Server, A “Main Server Configuration” - configuration options dedicated to the operation of our primary or main website and A “Virtual Host Configuration” section in which we can apply the same configuration options available in the Main Server Configuration section to any Virtual Hosts.

It is within this first section, the Global Environment Configuration section, that the majority of configuration changes have taken place.



# Configuring Apache 2.0

- global environment configuration
  - The prefork MPM accepts the same directives as apache 1.3
    - StartServers
    - MinSpareServers
    - MaxSpareServers
    - MaxClients
    - MaxRequestsPerChild

The first, obvious, change is the numerous `<IfModule>` directives in relation to loading the proper directives for whatever MPM has been chosen. Since the configure script chose the Prefork module for our Linux systems, we'll concern ourselves with that for the moment. (Feel free to look at the other directives to see how they relate to the other available MPMs).

Since the Prefork processing model is the very same processing model as previous versions of Apache, nothing within the `<IfModule prefork.c>` container should be of surprise. We'll be discussing these - as well as the other MPMs - later, in relation to Fine Tuning Apache 2.0.

For more on this topic, refer to the following documentation on the Apache Software Foundation's website:

\* <http://httpd.apache.org/docs-2.0/mpm.html>



# Configuring Apache 2.0

- global environment configuration
  - the BindAddress and Port directives no longer exist
    - their functionality is now provided by a more flexible Listen and ServerName directives.

With Apache 1.3 the functionality of the BindAddress and Port directives overlapped considerably. This led to confusion in regards to which directive should be used when. Since, after all, one of the main goals of redesign Apache 2.0 was to alleviate this type of confusion these directives have been removed. In their place the Listen and ServerName directives have picked up any necessary functionality that may have been required of these two former directives.

Documentation on these directives can be found at:

- \* [http://httpd.apache.org/docs-2.0/mod/mpm\\_common.html#listen](http://httpd.apache.org/docs-2.0/mod/mpm_common.html#listen)
- \* <http://httpd.apache.org/docs-2.0/mod/core.html#servername>



# Configuring Apache 2.0

- global environment configuration
  - the AddModule and ClearModuleList directives no longer exist
  - the order of the LoadModule lines is no longer relevant

The `AddModule` and `ClearModuleList` directives, which ensured that modules could be enabled in the correct order, no longer exist. This is due to the fact that with Apache 2.0 modules know what order they are supposed to load in since they specify their own ordering, thanks to changes in the Apache API. Thus these directives have been eliminated.

Also of note, since the modules themselves direct their order for loading the ordering of the `LoadModule` directive, which does remain, is no longer relevant.



# Configuring Apache 2.0

- global environment configuration
  - gone missing
    - ServerType
    - AccessConfig
    - ResourceConfig

Other directives that have “gone missing” with Apache 2.0 include: ServerType, AccessConfig and ResourceConfig.

ServerType allowed for the election of running the Apache daemon as a stand alone or inetd server. But due to the relatively frequent traffic a web server handles it becomes an expensive operation to use the inetd option and the Apache Group has strongly recommended against using this option. Moreover, in theory with Apache 2.0, this is an option, to run as a stand alone server or an inetd, that should be handled by an MPM. However, there is currently no MPM designed to be handle Apache as an inetd server.

As with ServerType. the AccessConfig and ResourceConfig directives have been included with Apache for various versions, but their functionality has been largely irrelevant. To include external configuration files one should use the Include directive. If you wish to ensure that the files are read in the order implied by the older directives the Include directives can be placed at the end of httpd.conf, with the one corresponding to ResourceConfig preceding the one corresponding to AccessConfig.

Documentation on these directives can be found at:

\* <http://httpd.apache.org/docs-2.0/mod/core.html#include>



# Configuring Apache 2.0

- main server configuration
  - the section has seen little change between apache 1.3 and 2.0
  - this is not by chance, but design

In order to facilitate in the migration from Apache 1.3 to 2.0 the Apache Group looked to minimize the changes that have taken place to the Main Server Configuration section. This is not to be read as meaning the Apache Group did not change the underlying functionality or code, but simply to mean the group tried to keep from complicating any migration with a round of completely new directives and syntax.



# Configuring Apache 2.0

- main server configuration
  - logging
    - the following logging directives have been removed
      - AgentLog
      - RefererLog
      - RefererIgnore
    - agent and referrer logs are still available using the CustomLog and LogFormat directives

As previously discussed, the `mod_log_referer` and `mod_log_agent` modules have been removed in Apache 2.0, totally superceded by `mod_log_config`. As such the directives, `AgentLog`, `RefererLog` and `RefererIgnore` have been removed as configuration options.

In replace of these directives one can use the `LogFormat` and `CustomLog` directives. For example to replace the functionality of the `RefererLog` directive use this combination of `LogFormat` and `CustomLog`:

```
LogFormat "%{Referer}I -. %U" referer
CustomLog logs/referer_log referer
```

Documentation on these directives can be found at:

- \* [http://httpd.apache.org/docs-2.0/mod/mod\\_log\\_config.html#customlog](http://httpd.apache.org/docs-2.0/mod/mod_log_config.html#customlog)
- \* [http://httpd.apache.org/docs-2.0/mod/mod\\_log\\_config.html#logformat](http://httpd.apache.org/docs-2.0/mod/mod_log_config.html#logformat)



# Configuring Apache 2.0

- main server configuration
  - directory indexing
    - FancyIndexing directive has now been removed
    - the new VersionSort option to the IndexOptions directive causes files containing version numbers to be sorted in a more natural way
    - the defaults for the ReadmeName and HeaderName directives have changed from README and HEADER to README.html and HEADER.html

The FancyIndexing directive has been demoted to an option for the IndexOptions directive. Another new option for IndexOptions, VersionSort, allows for a cleaner sorting of files containing version numbers and the default options for the ReadmeName and HeaderName directives have been changed from README and HEADER to README.html and HEADER.html.

Documentation on these directives can be found at:

- \* [http://httpd.apache.org/docs-2.0/mod/mod\\_autoindex.html#indexoptions](http://httpd.apache.org/docs-2.0/mod/mod_autoindex.html#indexoptions)
- \* [http://httpd.apache.org/docs-2.0/mod/mod\\_autoindex.html#readmeName](http://httpd.apache.org/docs-2.0/mod/mod_autoindex.html#readmeName)
- \* [http://httpd.apache.org/docs-2.0/mod/mod\\_autoindex.html#headerName](http://httpd.apache.org/docs-2.0/mod/mod_autoindex.html#headerName)



# Configuring Apache 2.0

- main server configuration
  - content negotiation
    - the CacheNegotiatedDocs directive now takes an argument on/off

CacheNegotiatedDocs now takes an argument: on | off Thus existing instances of CacheNegotiatedDocs should be replaced with CacheNegotiatedDocs on.

Documentation on this directive can be found at:

\* [http://httpd.apache.org/docs-2.0/mod/mod\\_negotiation.html#cachenegotiateddocs](http://httpd.apache.org/docs-2.0/mod/mod_negotiation.html#cachenegotiateddocs)



# Configuring Apache 2.0

- main server configuration
  - error documents
    - to use a hard-coded message with the `ErrorDocument` directive, the message should be enclosed in a pair of double quotation marks
    - greater flexibility with error documents

A basic change to the ErrorDocument directive now requires that any text included with the ErrorDocument directive in the configuration file must be enclosed with double quotation marks.

Of more interest, to most, is the greater flexibility of the ErrorDocument directive. For example, the ErrorDocument directive in combination with content negotiation and server side includes can be customized to serve error documents in the preferred language of a client, if specified. Examples of this internationalizing of error messages can be found in the Apache configuration file.

Documentation on this directive can be found at:

\* <http://httpd.apache.org/docs-2.0/mod/core.html#errordocument>



# Configuring Apache 2.0

- virtual hosts configuration
  - the contents of all `<VirtualHost>` containers should be migrated in the same way as the main server section

Since the Virtual Host section is simply a reimplementaion of the directives found in the Main Server section, only directed in relation to specific Virtual Hosts within a VirtualHost container it should be noted that the changes previous discusses as related to the Main Server also affect any Virtual Host configurations.

Documentation on this directive can be found at:

\* <http://httpd.apache.org/docs-2.0/vhosts/>



# Configuring Apache 2.0

- covalent script:
  - `http://apache.covalent.net/tools/index.php`
  - example conversion of an “in production” apache 1.3.27 `httpd.conf` file

Of final note, in relation to Configuration and Migration, Covalent has developed and released a Perl script that can assist in starting the migration process by making the basic changes previously discussed to an Apache 1.3 configuration file.

As an example of this script I've brought with me a "cleansed" - for the sake of security and privacy - configuration file for a production Apache 1.3 server for a demonstration.



# Modules Galore 2.0

“any sufficiently advanced  
technology is  
indistinguishable from magic”

-Arthur C. Clarke

Besides the resources already cited, the section on build PHP with Apache 2.0 comes, in part, from Dan Anderson's "Apache 2.0 and PHP (mod\_php) on Linux" which can be found at: <http://dan.drydog.com/apache2php.html>.



# Modules Galore 2.0

- hooks and optional functions
  - a hook represents an event that occurs in the course a connection
  - modules can register to participate in various hooks, indicating that they wish to have callback functions run during a specific stage of an http request

As introduced previously, with Apache 2.0 the API has been rewritten to allow greater flexibility for Apache's chief architectural feature, modules. Of the changes to the Apache API the first set we'll discuss cover changes to the traditional concept of Apache modules as handler modules - providing for an action based on a request or event, usual a HTTP request or event.

For handler modules we have a concept of a "hook" a representation of an event that can occur during a connection between a client and our web server. For a module to participate at a specific point during the connection the module simply registers to participate with a specific hook, indicating that the module wishes to have a function invoked once the proper connection event has been reached.



# Modules Galore 2.0

- hooks and optional functions
  - optional functions allow a module to register a function for use by another module with the core
  - if another module is interested in one of the registered functions, it checks with the core, retrieving a pointer to it if available

However, the real advantage with the Apache 2.0 API is that of Optional Functions. Optional Functions are functions that a module can registers such that other modules, which maybe interested in taking advantage of the provided function, can use.



# Modules Galore 2.0

- hooks and optional functions
  - the benefits of this system are nicely illustrated in `mod_include`, which handles all of its SSI tags through an optional function

An example of this Optional Function is provided within Apache 2.0 with `mod_include`. `mod_include` registers an Optional Function, which handles all processing of Server Side Include tags. Thus any module that wishes to define a new SSI tag can simply register the tag and a function to handle that tag with `mod_include`. This is done when the module that is supplying the new tag and function, looks up `mod_include`'s optional function for tag registration.



# Modules Galore 2.0

- filters
  - chain actions on input or output, not connection events

A new concept for Apache 2.0 is that of I/O filtering. A Filter is an action that is being applied to data that is sent or received by the server. That is a module, which is acting on a connection event, can apply a filter to act on the data within the connection. Multiple filters can be applied to the data, and the order of the filters can be explicitly specified.



# Modules Galore 2.0

- filters

- example: Common Gateway Interface (CGI) scripts can generate server-parsed HTML documents which can then be processed by `mod_include`
- the way this works is that each request is served by exactly one handler module followed by zero or more filter modules

As previously mentioned with Apache 1.3 we can invoke a CGI from a Server Side Include token, but the token needs to reside within a static document that our Apache server knows to process with `mod_include`. We cannot, in this given model, send a SSI token from a CGI script to invoke the SSI-based action. This is due to the fact that all modules, before Apache 2.0, write directly to the TCP stream. Thus we only get “one shot” a generating dynamic content - an all or nothing approach that does not provide an administrator with great flexibility. However, with Apache 2.0 we can invoke `mod_include` to process SSI tokens embedded in dynamic content. This is due to the reimplementaion of `mod_include` from a handler to a filter module.

An example configuration would be:

```
Options +ExecCGI
AddHandler cgi-script cgi
AddOutPutFilter INCLUDES cgi
```

Note: this change has ramifications if the `PATH_INFO` directive is used for a document which is handled by a module that is now implemented as a filter, as each contains trailing path information after the true file name. The core module, which initially handles the request, does not by default understand `PATH_INFO` and will return 404 Not Found errors for requests that contain such information. As an alternative, use the `AcceptPathInfo` directive to coerce the core module into accepting requests with `PATH_INFO`.

The following is an example of this directive:

```
AcceptPathInfo on
```

Documentation on these directives can be found at:

- \* <http://httpd.apache.org/docs-2.0/mod/core.html#acceptpathinfo>
- \* <http://httpd.apache.org/docs-2.0/handler.html>
- \* <http://httpd.apache.org/docs-2.0/filter.html>



# Modules Galore 2.0

- beginning with apache 2.0.42 the API will be kept stable
- third party modules, php example
  - `./configure --with-apxs2=/usr/local/apache-2.0.48/bin/apxs --prefix=/usr/local/apache-2.0.48/php-4.3.4 --disable-cgi`

But of immediate importance to most of us in migrating from Apache 1.3 to 2.0 is that of third party modules that have been ported to the new API. Most, no doubt, are aware of the issues of bring PHP to 2.0's API. In part this was do to a changing API even after the first "general availability release". However, as noted the Apache Group has since dubbed Apache 2.0 stable and all new development now occurs within the 2.1 branch.

So let us partake in adding PHP as a Dynamic module to Apache 2.0. For this we'll need the latest release of PHP, as of nov 12th, 2003 this was 4.3.4. After unpacking our source code the minimum invocation of the configure script we need is:

```
./configure --with-apxs2=/usr/local/apache-2.0.48/bin/apxs --  
prefix=/usr/local/apache-2.0.48/php-4.3.4 --disable-cgi
```

If one chooses to use PHP as an Apache 2.0 Filter then one changes the configure parameters when configuring to PHP from --with-apxs2 to --with-apxs2filter.

Documentation of installing PHP 4.3.x with Apache 2.0.x can be found on the PHP website at: <<http://us4.php.net/manual/en/install.apache2.php>>



# Intermission

- 15 minutes while everyone's laptops run the PHP build processes

So have at it ;-)



# Modules Galore 2.0

- adding in PHP
  - `LoadModule php4_module`  
`modules/libphp4.so`
  - `DirectoryIndex index.html index.php`
  - `AddType application/x-httpd-php php`
- test PHP installation

Once we have a working module all that is left for us to do is add in our configuration options. If we built PHP as a filter then we need to use the Filter related directives:

```
<Files *.php>  
    SetOutputFilter PHP  
    SetInputFilter PHP  
</Files>
```

Documentation on these directives can be found at:

- \* <http://httpd.apache.org/docs-2.0/mod/core.html#setinputfilter>
- \* <http://httpd.apache.org/docs-2.0/mod/core.html#setoutputfilter>



# Fine Tuning Apache 2.0

“perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away”

-Antoine de Sain

The following references provide invaluable information for this section of the lab:  
Apache Server Bible by Mohammed J. Kabir, ISBN# 0764532189  
Professional Apache 2.0 by Peter Wainwright, ISBN# 1861007221



# Fine Tuning Apache 2.0

- MPM
  - best linux option: prefork
    - too early
      - third party modules, the source of apache's success, may not be threads aware yet
      - processing model is understood
    - other MPMs options in future
    - threading probably won't provide large benefit for most linux systems anyway

Of all the options for Apache 2.0 on Linux the first and foremost question is what processing model to use? At least for now, the best option is still Prefork.

First, as modules from 1.3 port themselves to 2.0, out of the box they'll be able to handle the Prefork model. For example, only parts of PHP or mod\_perl are threads aware. It'll probably be some time till the majority of these useful tools are threads aware. Moreover, in terms of migration the same directives, such as MinSpareServers, MaxSpareServers and MaxRequestsPerChild are in use for optimizing Apache performance.

The MPMs currently available are only the tip of the iceberg, an introduction to what Apache 2.0 can do. As the Apache Group and others start to take advantage of this new ability we will, hopefully, see new and improved models for Apache 2.0 on all platforms, including Linux.

Threading also adds a bit of instability, even with the Apache Group's hybrid solution. While this may not be a big issue for most, those who need to worry about high availability of the web server, in conjunction with or superceding the scalability of the server, this is an important issue not to switch processing models.

Finally, adding threaded processes in the long run probably won't reap major benefits for most web servers running Apache on Linux. As we discussed previously the advantage of MPMs really comes in use when dealing with Apache scalability in relation to portability.



# Fine Tuning Apache 2.0

- apache
  - keep an eye on invoking resource intensive configurations
    - HostNameLookup
    - FollowSysLinks
    - KeepAlive, KeepAliveTimeout, MaxKeepAlive
    - Log level option
    - session tracking
    - .htaccess files, large httpd.conf files
    - modules such as
      - mod\_status
      - mod\_rewrite

As with previous version of Apache there are other configuration options that can change the performance of the server, no matter what MPM is chosen. Some of these directives and options include:

**Timeout** - This directive "Timeout" is used to define the amount of time Apache will wait for a GET, POST, PUT requests and acknowledgements.

**KeepAlive** - KeepAlive enables persistent connections on the web server. For better performance, it's recommended to set this option to "On" and allow more than one request per connection.

**MaxKeepAliveRequests** - This directive is used to define the number of requests allowed per connection when the KeepAlive option above is set to "On".

**KeepAliveTimeout** - is used to define how much time, in seconds, Apache will wait for a subsequent request before closing a connection.

**HostnameLookups** - This directive specifies DNS lookups. It's recommended to set this option to "Off" in order to avoid latency with every request.

Documentation on these directives in relation to fine tuning can be found at:

\* <http://httpd.apache.org/docs-2.0/misc/perf-tuning.html>



# Fine Tuning Apache 2.0

- linux
  - kernel
    - handle number processes/threads
  - filesystem
    - optimize block size
    - enable journaling

What kernel and filesystem is being used on our Linux system is just as important.

For example, the Linux 2.4 series kernel provides for a large number of simultaneous processes or threads - allowing greater scalability. As with the Apache server the Linux kernel also provides a configurable process limit. The threading model used in Linux 2.4 has been changed to a scalable or "soft" version. Previously the thread limit was 1024, which led to poor performance with large numbers of users. The limit is now set at run time.

The Second Extended filesystem is widely used within the Linux operating system. Depending on the usage of our filesystem we can optimize the block size. Blocks can be 1024, 2048, or 4096 bytes in size. With larger block sizes, fewer disk accesses are needed as more info and data can be read at a time, which increases the file system's performance. The drawback with larger blocks means more space may be wasted in the final block of a file, so there is a storage space hit associated with them.

Another option in relation to the Linux filesystem is that of journaling. A journaling filesystem is a filesystem that maintains a log (or journal), the contents of which are not cached. Whenever the filesystem is updated, a record describing the transaction is added to the log. If the machine crashes, the background process is run on reboot and simply finishes copying updates from the journal to the filesystem. Incomplete transactions in the journal file are discarded, so the filesystem's internal consistency is guaranteed.



# Fine Tuning Apache 2.0

- apache
  - use apache bench for testing
    - `./ab -n5000 -c 500 -k localhost/`
      - 5,000 request, 500 at a time, using KeepAlive requests to our local server

Apache Bench is a handy little command-line tool included in the distribution of Apache. With it one can test the performance of a server by specifying a number of requests to send to the server individually or concurrently. If concurrency isn't specified, Apache Bench sends a request, awaits the response, sends the following one, awaits the response, and so on until the sending of all the requests.

As an example let us take our freshly built Apache out for a test drive.

Documentation on is tool can be found at:

\* <http://httpd.apache.org/docs-2.0/programs/ab.html>



# The Road Ahead

“for a successful technology,  
reality must take precedence  
over public relations, for  
nature cannot be fooled”

-Richard Feynman

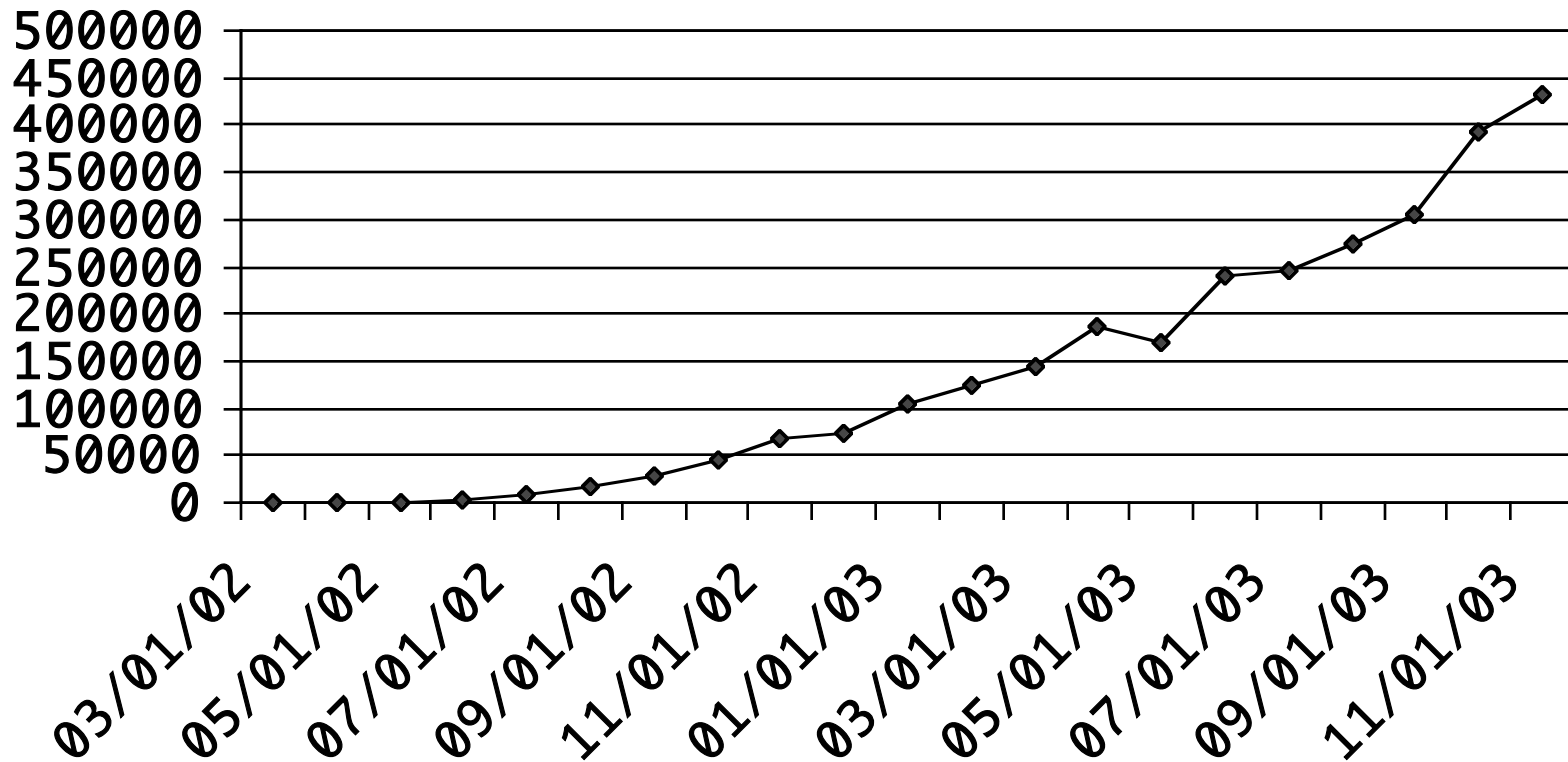
Finally, let us take a step back and see what is ahead for the future of Apache and the Apache Group.

The basis for the information included within this section includes Security Space, <<http://www.securityspace.com>> and ServerWatch, <<http://www.serverwatch.com>>.



# The Road Ahead

usage of apache 2.0.x



## Growth of Apache 2.0 adoption since the first general availability release in April 2002

### Notes about data:

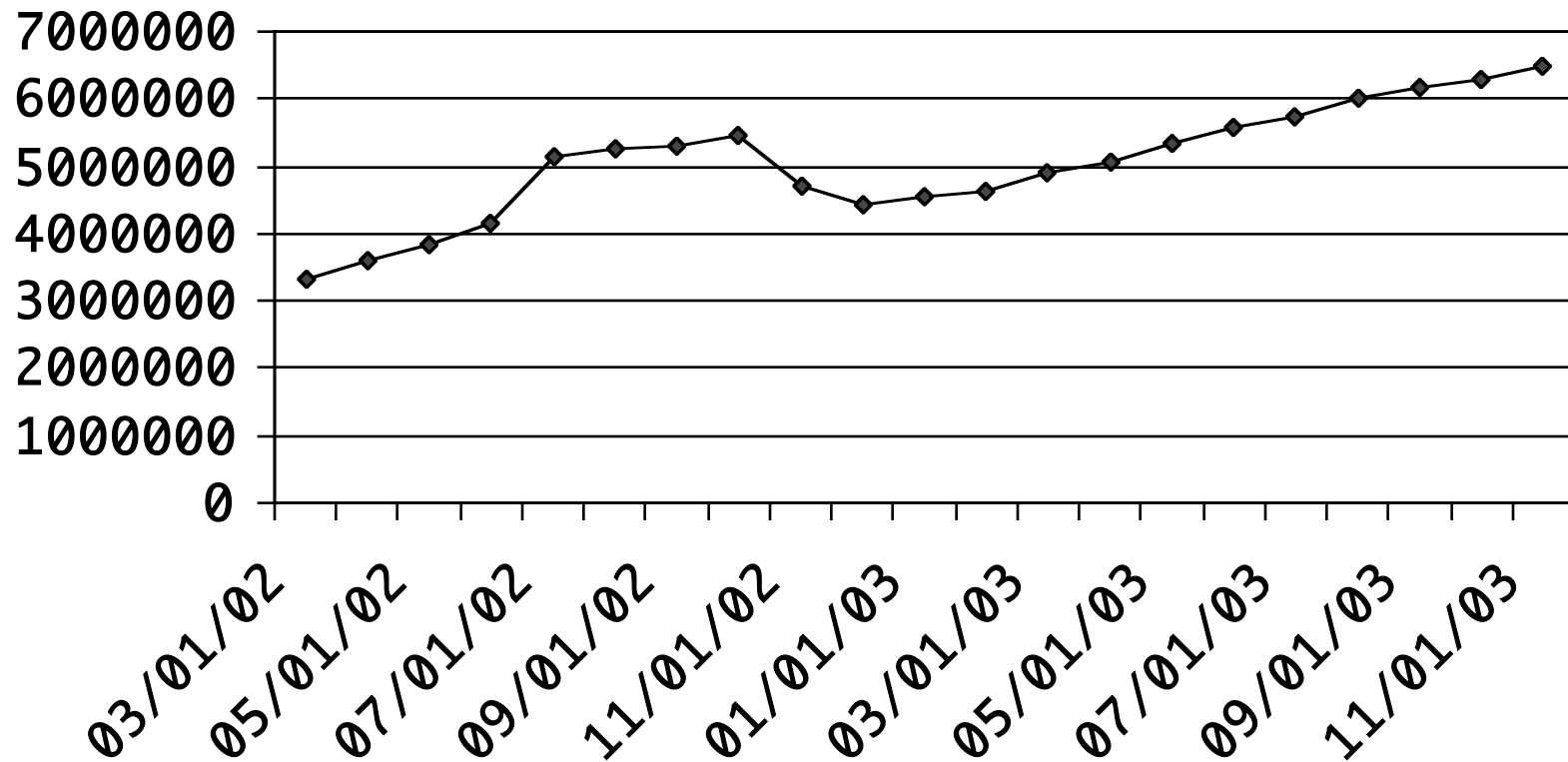
\* Results supplied by <http://www.securityspace.com> and include versions of Apache that responded with versions number in their header for all domains queried

\* These totals do not include servers "powered by Apache" such as web servers supplied by IBM, Covalent or Oracle



# The Road Ahead

usage of apache 1.3.x



In comparison use of Apache 1.3 for the same time frame.



# The Road Ahead

- possible road blocks to adoption:
  - lack of support
    - hosting companies, modules
    - “if it ain't broke don't fix it”

No doubt one can offer a number of explanations for the slow adoption of Apache 2.0. Lack of full support from Linux and other Un\*x platforms, lack of information or migration options.

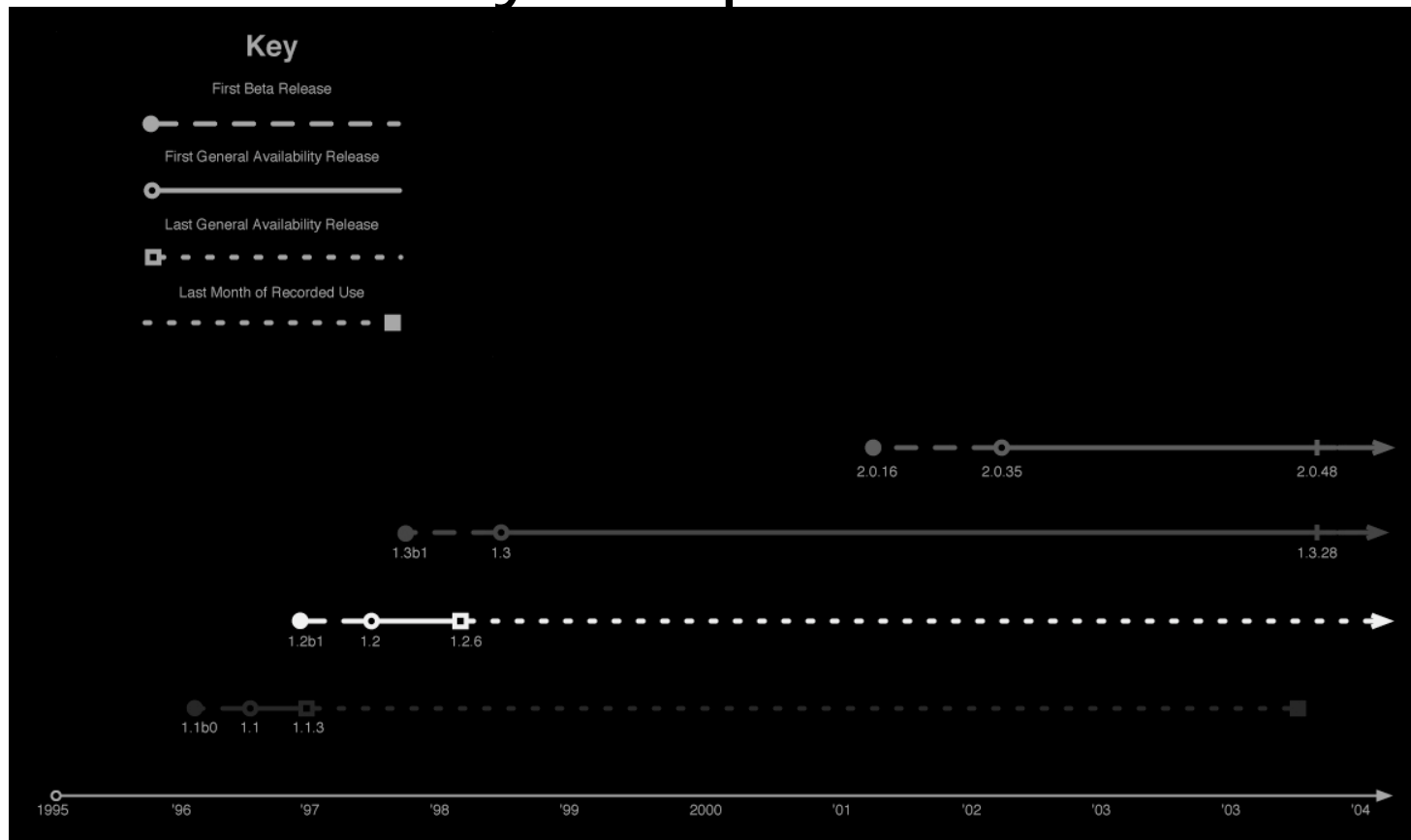
Of these, the one that must catch our eye is that of third-party modules. Third party modules, such as PHP, are the linchpin for Apache. Of these modules, the changing API for Apache 2.0, even after its general release, keep third-party developers at bay. As one Apache developer noted for IT Week, <http://www.computing.vnunet.com/News/1134850>, “Most third-party module authors are not willing to maintain and change their code for every Apache 2 release; sometimes the APIs were changed two or three times within one development cycle.”

For example the PHP Documentation still states: “Do not use Apache 2.0 and PHP in a production environment neither on Unix nor on Windows.” This probably won’t change till PHP 5, which is currently in beta testing, is released.



# The Road Ahead

## history of apache releases



In fact the stats on Apache Web Server deployment show that only recently, Oct of 2003, did the last installations of Apache 1.1.x migrate to a different web server. The current rate of removal for Apache 1.2.x shows that this version will finally leave the Internet around February 2005 (Nov 2003 survey shows 27150 domains running some version of 1.2.x series of releases with an average decline of 1724.8 domains migrating to a different web server over the last 15 months). Whereas Apache 1.3.x, as we have seen, has yet to show an continuous decline. In fact in May of 2000 Apache Week did its own study on the migration of Apache versions, noting that “Tak[ing] into account the number of sites using Apache ... the actual number of sites using older releases ... rise[s] for anything up to three months after a new release becomes available.” However, these studies cannot show if the migrations patterns are “attributable to upgraders or new adopters.”

While Apache Week was looking at specific release points instead of overall code branches, the long lifespan of any given Apache branch does give us something to speculate on. That each code base sees a group of early adopters that is willing to test their websites(s) using the latest code branch. This is the current state of the Apache 2.0 branch. Unlike the early adopters, the next wave of migrations won't come till the stability and resourcefulness of the new code base has been proven. Once done, hosting companies and other corporations will then migrate their critical applications and systems to the new branch. This migration represents the bulk users for any branch of Apache. After the majority of these “critical migrations”, the remain non-critical deployments are migrated as time permits.



# The Road Ahead

- what's next for apache?
  - 2.1 developments
    - redesign of the MPM code
    - redesign of AAA  
(authentication/authorization/access control) code

As noted, the Apache 2.0 internals have gone through a major overhaul, yet one will probably not see an overwhelmingly huge performance boost. The Apache Group has already starting the development of the next stable Apache Web server release, 2.2. For this release the development branch of 2.1 will probably see changes for tightly tuning performance.

It should be noted that while 2.0 doesn't bring a greater efficient its performance hasn't been degraded by the code rewrite either. Apache 2.0 performance is on par with Apache 1.3. In terms of real benefit Apache 2.0 has greater flexibility such as I/O filtering and easier to manage configuration file.

Another possible change for Apache may come in the reimplementation of the Authentication Authorization and Access Control code. As Rich Bowen outlined in his article "Safer Apache Driving with AAA" for ServerWatch, <<http://www.serverwatch.com/tutorials/article.php/2202671>>, "In Apache 1.3, and Apache 2.0, these three processes tend to get slightly jumbled together .. [whereas] ... in the Apache 2.1 AAA framework, these things are more clearly separated. This is primarily to the benefit of module developers, but also helps the server administrator to have an enormous amount of additional control over how things happen."

The first Alpha release for Apache 2.1 will no doubt happen sometime within the next 6 - 12 months.



# Review

- what was covered
  - what is apache 2.0
  - building, configuring
  - running, troubleshooting





# This Presentation

- `<http://www.weinstein.org/work/presentations/linuxworld/apache2ny/>` (HTML)
- `<http://www.weinstein.org/work/presentations/linuxworld/apache2ny.pdf>` (PDF)

