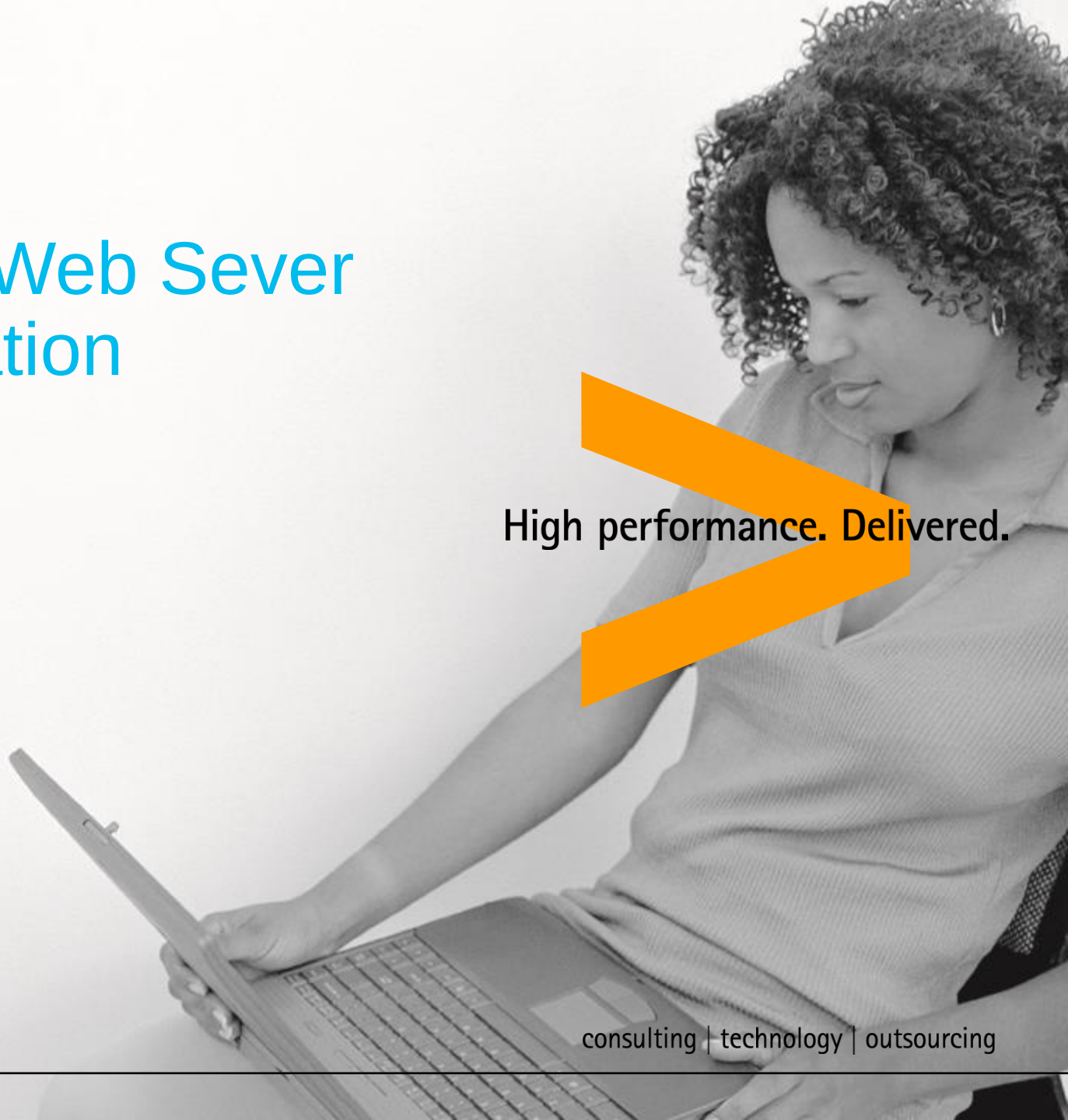
Apache 2 Web Sever Administration



consulting | technology | outsourcing

Apache 2 Administration

Goals

- Introduction to Apache
- Installing Apache
- Managing Apache
- Modules & directives
- Configuring Apache
- Security
- Secure sockets layer (ssl) security

Apache 2 Administration

Goals

- Logs
- Dynamic web content
- Virtual hosts, redirection and indexing
- Redirection methods
- Enhance or limit functionality using modules
- Directory indexing
- Proxy servers and firewalls

Apache 2 Administration

Goals

- Monitoring apache web server
- Content Caching
- URL Mapping
- Apache web server status & server information
- Performance Tuning
- Improving performance

Apache 2 Administration

Goals

- Apache virtual host workflow and configuration in detail
- Apache MPM configurations and usages
- Server capacity (like RAM, ROM) needed for n number of users (say n=3000+).
- Apache tuning when there is high server load
- Other Apache mechanism for handling uncertain situations

APACHE
HTTP SERVER

The name Apache Server has been taken from Native American tribe '*Apache*', famous for its skills in warfare and strategy making

Apache Web Server (HTTP)

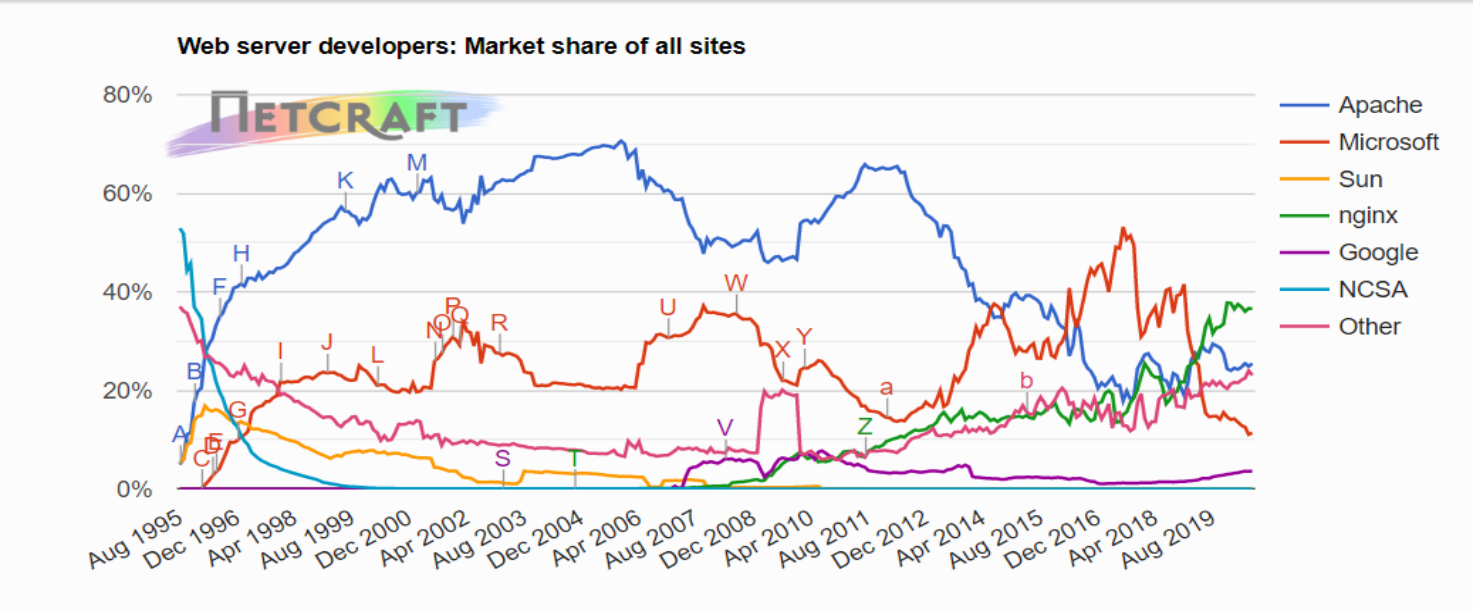
Apache is the most widely used web server software. Developed and maintained by Apache Software Foundation

Introduction To Apache 2

- In the July 2020 survey Netcraft survey received responses from 1,234,228,567 sites across 260,658,118 unique domains and 10,221,919 web-facing computers.
- This represents a gain of 9.47 million sites and 180,000 computers, but a loss of 1.75 million domains.
- July 2020 most of the major server vendors saw gains in total sites this month: Apache gained 9.8 million sites.
- While Microsoft and nginx gained 5.4 million and 2.5 million sites respectively.
- LiteSpeed continued to see strong growth, gaining 1.95 million new sites this month.

Introduction To Apache 2

Developer	June 2020	Percent	July 2020	Percent	Change
nginx	448,673,487	36.63%	451,156,878	36.55%	-0.08
Apache	304,288,405	24.84%	314,054,523	25.45%	0.60
Microsoft	134,874,928	11.01%	140,264,332	11.36%	0.35
Google	43,449,240	3.55%	44,290,430	3.59%	0.04



Who is using Apache Web Server?

- Popular Companies Using Apache HTTP Server:
  - IBM
  - eBay
  - Adobe
  - PayPal
  - Linkedin
  - GTMetrix
  - Facebook
  - **Aspire**

Alternative to Apache HTTP Server

- Nginx
- Apache Tomcat
- Node.js
- Lighttpd
- Cherokee
- Microsoft IIS
- Appweb
- Hiawatha

Apache Rocks On

- Apache is a highly configurable Web Server with a modular design.
  - It is very easy to extend the capabilities of Apache Web server. Anyone with decent C or Perl programming expertise can write a module to perform a special function. This means that there are tons of Apache modules available for use.
- Apache is a free, open source technology.
- Apache works great with Perl, PHP, and other scripting languages.

Apache Rocks On

- Apache runs on Linux and other Unix systems.
- Apache also runs on Windows.

Apache: The Beginning

- In the early days of the Web, the National Center for Super Computing Applications (NCSA) created a Web server that became the number one Web server in early 1995.
- However, the primary developer of the NCSA Web server left NCSA about the same time, and the server project began to stall.
- In the meantime, people who were using the NCSA Web server began to exchange their own patches for the server and soon realized that a forum to manage the patches was necessary.
- **The Apache Group was born**. The group used the NCSA Web server code and gave birth to a new Web server called Apache.

Apache: The Beginning

- Originally derived from the core code of the NCSA Web server and a bunch of patches, the Apache server is now the talk of the Web server community.
- In three short years, it acquired the lead server role in the market.

Apache: The Beginning

- **The very first version (0.6.2)** of publicly distributed Apache was released in April 1995.
- The 1.0 version was released on December 1, 1995.
- The Apache Group has expanded and incorporated as a nonprofit group.
- The group operates entirely via the Internet.
- However, the development of the Apache server is not limited in any way by the group.
- Anyone who has the know-how to participate in the development of the server or its component modules is welcome to do so, although the group is the final authority on what gets included in the standard distribution of what is known as the Apache Web server.

Apache: The Beginning

- This allows literally thousands of developers around the world to come up with new features, bug fixes, ports to new platforms, and more.
- When new code is submitted to the Apache Group, the group members investigate the details, perform tests, and do quality control checks. If they are satisfied, the code is integrated into the main Apache distribution.

What Is Apache?

Supports virtual hosting

Allows you to implement highly dynamic coding

A popular choice among web developers

Compatible with Windows, macOS, Unix

A process-based, modular, open-source web server application

Apache vs IIS

Feature	IIS	Apache
Supported OS	Windows	Linux, Unix, Windows, Mac OS
User support & fixes	Corporate support	Community Support
Cost	Free, but bundled with Windows	Completely free
Development	Closed, proprietary	Open source
Security	Excellent	Good
Performance	Good	Good
Market Share	32%	42%

Current Apache Versions

Versions of Apache HTTP Server

Version	Initial release	Latest release
1.3	1998-06-06[54]	2010-02-03 (1.3.42)[55]
2.0	2002-04-06[56]	2013-07-10 (2.0.65)[57]
2.2	2005-12-01[58]	2017-07-11 (2.2.34)[59]
2.4	2012-02-21[60]	2020-04-01 (2.4.43)[61]

Old version **Latest version**

The Apache Feature List

- One of the greatest features that Apache offers is that it runs on virtually all widely used computer platforms.
- At the beginning, Apache used to be primarily a Unix based Web server, but that is no longer true.
- Apache not only runs on most flavors of Unix, but it also runs on Windows 2000/NT/9x and many other desktop and server-class operating systems such as Amiga OS 3.x and OS/2.
- **Apache HTTP Server versions later than 2.2 will not run on any operating system earlier than Windows 2000.**

The Apache Feature List

- Apache offers many other features including
  - fancy directory indexing
  - directory aliasing
  - content negotiations
  - configurable HTTP error reporting
  - SetUID execution of CGI Programs
  - resource management for child processes
  - server-side image maps
  - URL rewriting
  - URL spell checking
  - online manuals.

The Apache Feature List

- Apache offers many other features including
  - Simple, yet powerful file-based configuration
  - Support for CGI (Common Gateway Interface)(mod_cgi)
  - Support for FastCGI(mod_fcgi)
  - Support for virtual hosts
  - Support for HTTP authentication
  - Integrated Perl(mod_perl)
  - Support for PHP scripting(mod_php)
  - Java Servlet support(jsp)

The Apache Feature List

- Apache offers many other features including
  - Handling of static files
  - Loadable dynamic modules
  - Auto-indexing
  - .htaccess
  - Compatible with IPv6
  - Supports HTTP/2
  - FTP connections
  - Gzip compression and decompression
  - Bandwidth throttling
  - Perl, PHP, Lua scripts
  - Load balancing
  - Session tracking
  - Geolocation based on IP address

The Apache Feature List

- Apache offers many other features including
  - Integrated Proxy server
  - Server status and customizable logs
  - Support for Server-Side Includes (SSI)
  - Support for Secured Socket Layer (SSL)(mod_ssl)

Fancy Indexing

- Fancy indexing shows the files plus additional information about them like the date the file is last modified, their size and description.
- To use that type of indexing, add the following two lines in a file named .htaccess in the destination folder:
  - Options +Indexes
  - IndexOptions +FancyIndexing

Directory Alias

- Directory alias are used to map between URLS entered in the browser to filesystem paths.
- Creating an alias on the installation server enables the URL: http://servername/oss114 to map to the directory located at /install/11-4 and solving my problem.

Content Negotiation

- Content negotiation is the ability of a web server to deliver the document that best matches the browser's preferences/capabilities.
- For example, if a resource exists in multiple languages, the web server can choose which variant it serves based on the Accept-Language header delivered by the browser.

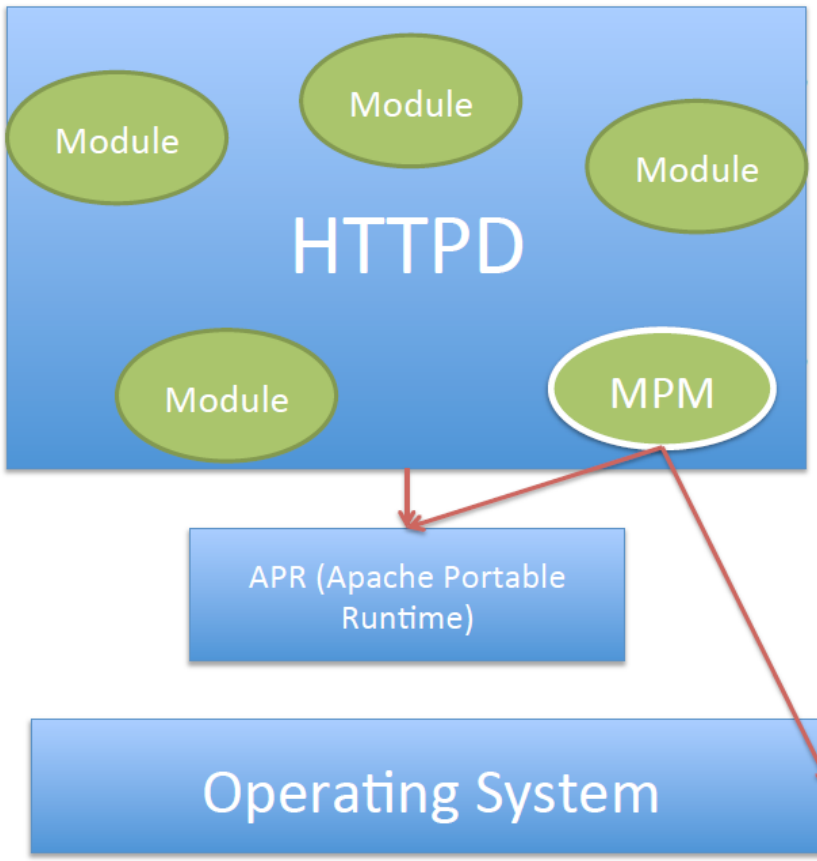
SetUID execution of CGI Programs;

- CGI root access should have SetUID

Virtual Host

- An Apache web server can host multiple websites on the SAME server.
- You do not need separate server machine and apache software for each website.
- This can be achieved using the concept of Virtual Host or VHost.
- Any domain that you want to host on your web server will have a separate entry in apache configuration file.

Apache 2 Architecture



small core

several modules

- compiled statically or loaded dynamically

Cross platform utilities (APR)

Apache 2.4 Architecture

Multiprocessing modules

- The first major change in Apache 2.0 is the introduction of multiprocessing modules (MPMs).
- To understand why MPMs are created, you need to understand how Apache worked before.
- **Apache Version 1.3 or earlier used a preforking architecture.**
- In this architecture, an Apache parent process forked a set of child processes, which serviced the actual requests.
- The parent process simply monitored the children and spawned or killed child processes based on the amount of requests received.
- Unfortunately, this model didn't work well under platforms that are not process-centric such as Windows.
- So, the Apache Group came up with the MPMbased solution.

Apache 2.4 Architecture

- Each MPM is responsible for starting the server processes and for servicing requests via child processes or threads depending on the MPM implementation.
- Several MPMs are available.

How Apache Works: MPM

- MultiProcessing Modules (MPMs) since Apache2:
  - In apache 1.3 uses a preforking architecture
    - the parent creates/destroys children if required
    - does not work well on some platform (such Windows)
  - MPM offers several alternatives (implemented in MPM modules) :
    - prefork MPM (like Apache 1.3)
    - worker MPM (multiple child, each one with several threads)
    - winnt MPM: single process, multithread (specific for windows)

BeOS	beos
Netware	mpm_netware
OS/2	mpmt_os2
Unix	prefork
Windows	mpm_winnt

```
<IfModule mpm_prefork_module>
    StartServers          5
    MinSpareServers       5
    MaxSpareServers      10
    MaxClients          150
    MaxRequestsPerChild   0
</IfModule>
```

We can tune parameters in /etc/apache2/apache2.conf

```
<IfModule mpm_worker_module>
    StartServers          2
    MaxClients          150
    MinSpareThreads      25
    MaxSpareThreads      75
    ThreadsPerChild      25
    MaxRequestsPerChild   0
</IfModule>
```

Apache 2.4 Architecture

- **The prefork MPM**
- The prefork MPM mimics the Apache 1.3 or earlier architecture, creating a pool of child processes to service requests.
- Each child process has a single thread.
- For example, if Apache starts 30 child processes, it can service 30 requests simultaneously.
- If something goes wrong and the child process dies, only a single request is lost.

Apache 2.4 Architecture

- The number of child processes is controlled using a minimum and maximum setting.
- When the number of requests increases, new child processes are added until the maximum is reached.
- Similarly, when the requests fall, any extra child processes are killed.

The threaded MPM

- This MPM enables thread support in Apache 2.0.
- This is like the prefork MPM, but instead of each child process having a single thread, each child process is allowed to have a specified number of threads.
- Each thread within a child process can service a different request.
- If Apache starts 30 child processes where each child is allowed to have at maximum 10 threads, then Apache can service 30 × 10 = 300 requests simultaneously.

The threaded MPM

- If something goes wrong with a thread, for example, an experimental module causes the thread to die, then the entire process dies.
- This means that all the requests being serviced by the threads within the child process will be lost.
- However, because requests are distributed among threads on separate child processes, it is likely that a child's death will take down at maximum of 1/n of all the total connection, where n presents the number of all simultaneous connections.

The threaded MPM

- A process is added or removed by monitoring its spare-thread count.
- For example, if a process has less than the minimum number of spare threads, a new process is added.
- Similarly, when a process has a maximum number of idle threads, it killed.
- All processes run under the same user and group ID assigned to Apache server.
- Because threads are more resource efficient than processes, this MPM is very scalable.

The perchild MPM

- This is also new in Apache 2.0.
- In this MPM model a set number of child processes are started with a specified number of threads.
- As request load increases the processes add new threads as needed.
- When request count reduces, processes shrink their thread counts using a minimum and maximum thread count setting.
- The key difference between this module and the threaded MPM is that the process count is static and also each process can run using a different user and group ID.

Apache 2.4 Architecture

The perchild MPM

- This is also new in Apache 2.0.
- In this MPM model a set number of child processes are started with a specified number of threads.
- As request load increases the processes add new threads as needed.
- When request count reduces, processes shrink their thread counts using a minimum and maximum thread count setting.
- The key difference between this module and the threaded MPM is that the process count is static and also each process can run using a different user and group ID.
- This makes it easy to run different virtual Web sites under different user and group IDs.

Apache 2.4 Architecture

The winnt MPM

- This is the MPM for the Windows platform, including Windows 2000, Windows NT, and Window 9x.
- It is a multithreaded module.
- Using this module Apache will create a parent process and a child process.
- The child process creates all the threads that services the request.
- Also, this module now takes advantage of some Windows-only native function calls, which allows it to perform better than the earlier versions of Apache server on Windows platform.

Apache 2.4 Architecture

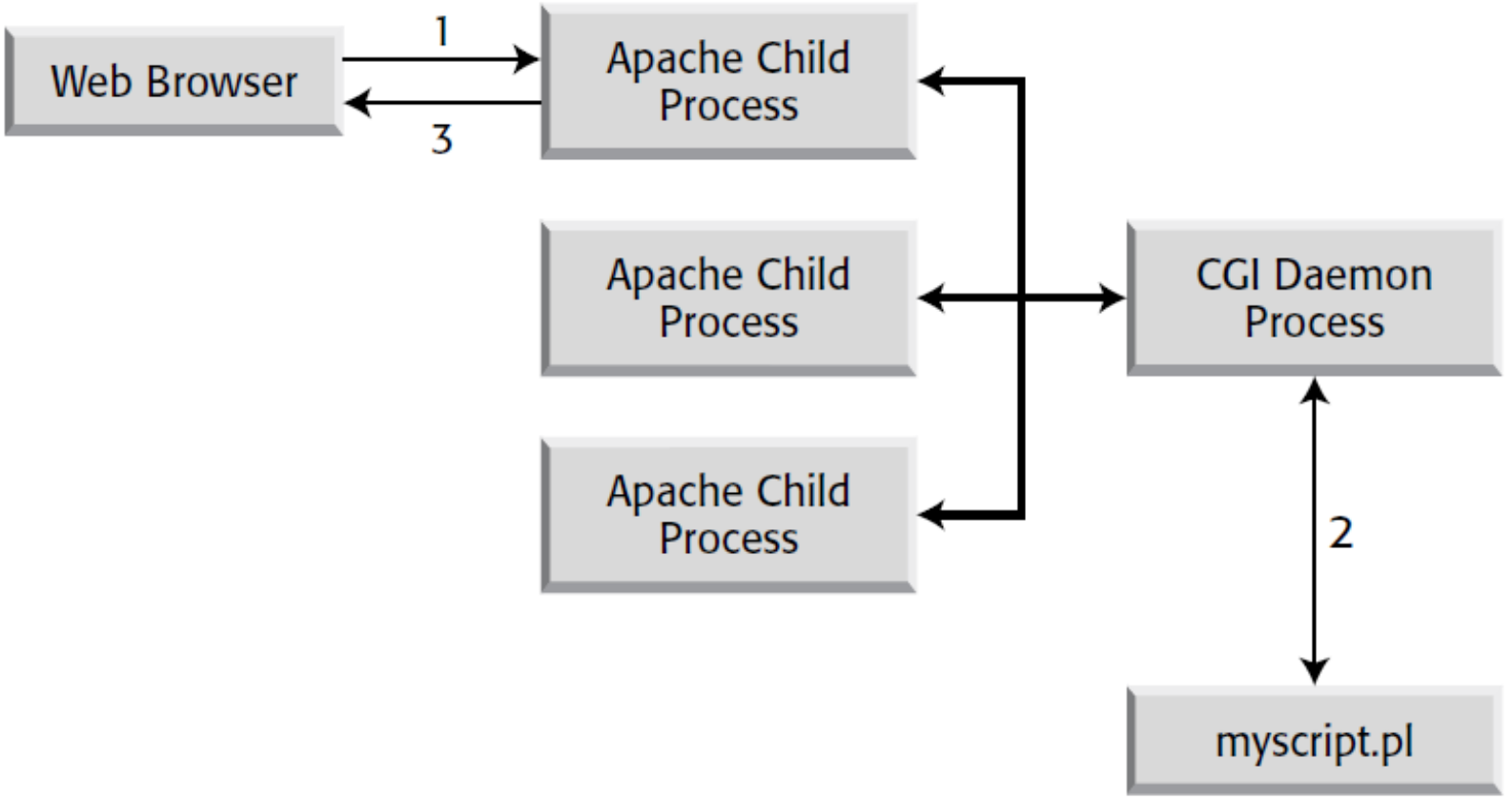
Filtering I/O

- Apache 2.0 now provides architecture for layered I/O.
- This means that one module's output can become another module's input.
- This filtering effect is very interesting.
- For example, the output produced by CGI scripts, which is processed by the mod_cgi module, can now be handed to the mod_include module responsible for SSIs.
- SSI (Server Side Includes) are directives that are placed in HTML pages, and evaluated on the server while the pages are being served.
- In other words, CGI scripts can produce output as SSI tags, which can be processed before the final output is sent to the Web browser.

New CGI daemon

- Because many of the MPM modules use threads, executing CGI scripts become cumbersome when a thread gets such a request.
- The mod_cgi module still works, but not optimally for threaded MPMs, so mod_cgid was added.
- The mod_cgid module creates a daemon process, which spawns CGI processes and interacts with threads more efficiently.
- Figure shows how a CGI request for a script called
- myscript.pl is serviced.

Apache 2.4 Architecture



Apache 2.4 Architecture

- Here is how the CGI scripts are executed:
- 1. When the CGI request comes to a thread within a child process, it passes the request to the CGI daemon.
- 2. The CGI daemon spawns the CGI script and outputs the CGI script-generated data to the thread in the child process.
- 3. The thread returns the data to the Web browser.

Apache 2.4 Architecture

- When the main Apache server starts, it also starts the CGI daemon and establishes a socket connection.
- So, when a new child process is created, it inherits the socket connection and therefore does not have any need to create a connection to the CGI daemon for each request.
- The entire process improves CGI execution in the threaded environment.

Apache 2.4 Architecture

Apache Portable Run-Time

- In furtherance of the Apache Group's vision to create the most popular Web server in the world, it became clear that Apache's portability needed to be addressed in Apache 2.0.
- Prior to the current release, Apache had been dealing with portability internally, which made the code base less manageable.
- So, Apache Group introduced the Apache Portable Run-Time (APR). APR's purpose is to provide a single C interface to platform-specific functions so that code can be written once.
- This enables Apache to run more natively on platforms such as Windows, BeOS, Amiga, and OS/2.
- Because of APR, many programs, such as ApacheBench, can run on these platforms.

Understanding the Apache License

- Free software such as Apache, Perl (Practical Extraction and Reporting Language), and Linux (an x86-based Unix clone operating system) are getting a great deal of press because of Netscape's decision to make Netscape Communicator, one of the most popular Web browsers, available for free with its Mozilla project.
- Unfortunately, free software such as Apache, Perl, and Linux do not share the same licensing agreements, and so the media has created some confusion by associating these packages in the same licensing category.

Understanding the Apache License

- All free software is intended to be, well, free for all.
- However, there are some legal restrictions that the individual software licenses enforce.
- For example, Linux, which is made free by GNU Public License (GPL), requires that any changes to Linux be made public.
- Apache, on the other hand, does not require that changes made to Apache be made public by anyone.

Understanding the Apache License

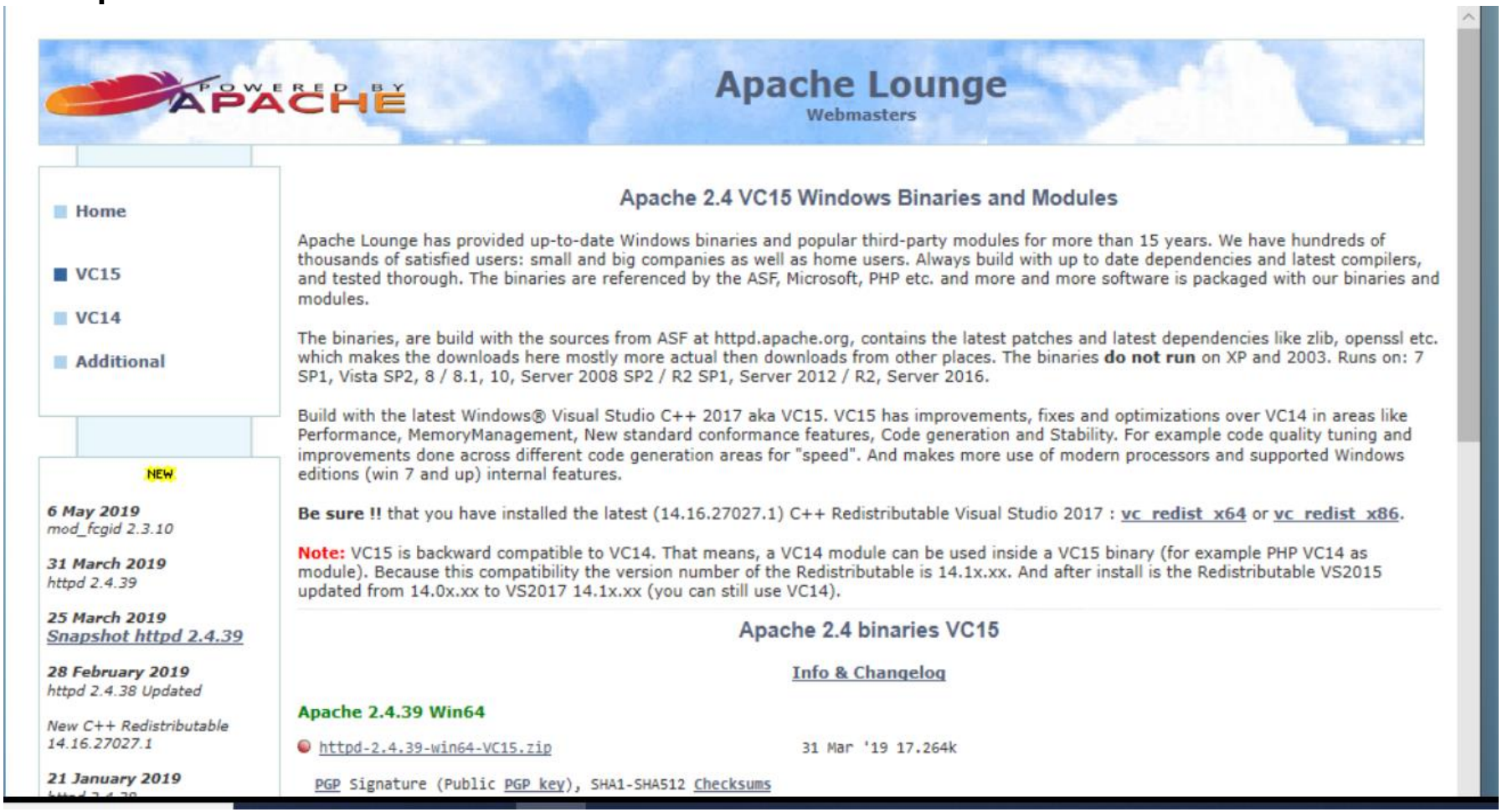
- In short, think of Apache as free, copyrighted software published by the Apache Group.
- It is neither in the public domain nor is it shareware.
- Also note that Apache is not covered by GPL.

How to install & configure Apache on a Windows server

- Step 1 - Download Apache for Windows
- Step 2 - Unzip
- Step 3 - Configure Apache
- Step 4 - Start Apache
- Step 5 - Check Apache
- Step 6 - Install Apache as a Windows service
- Step 7 - Monitor Apache (optional)

How to install & configure Apache on a Windows server

Step 1 - Download



How to install & configure Apache on a Windows server

- Step 2 - Unzip
  - Once the download has completed, open the downloaded "httpd-2.4.39-win64-VC15.zip" file, and extract its contents to a suitable location on your server i.e. C:\Apache24 or D:\Apache, etc

How to install & configure Apache on a Windows server

- **Step 3 - Configure Apache**
  - Once you've extracted Apache, configure it.
  - Start by locating the file "httpd.conf" (which will be in the "conf" sub directory), and open this in a standard text editor.
  - By default, this configuration file assumes that you've extracted Apache to C:\Apache24.
  - If however you've extracted Apache to a different location (i.e. D:\Apache), you'll need to update the ${SRVROOT} variable within in the httpd.conf file to point to the new location accordingly, i.e:
  - Define SRVROOT "C:/Apache24" → Define SRVROOT "D:/Apache"

How to install & configure Apache on a Windows server

- **Step 3 - Configure Apache**
  - If there is no SRVROOT variable present in your httpd.conf file, then instead we'll need to manually update all instances of "C:/Apache24", i.e.:
  - ServerRoot "C:/Apache24" → ServerRoot "D:/Apache"
  - DocumentRoot "C:/Apache24/htdocs" → DocumentRoot "D:/Apache/htdocs"
  - <Directory "C:/Apache24/htdocs"> → <Directory "D:/Apache/htdocs">
  - ScriptAlias /cgi-bin/ "C:/Apache24/cgi-bin/" → ScriptAlias /cgi-bin/ "D:/Apache/cgi-bin/"

How to install & configure Apache on a Windows server

- **Step 3 - Configure Apache**
  - Regardless of where you extracted Apache to, you'll also need to make the following modifications to the http.conf file:
  - A) Add "ExecCGI" to "Options" directive:
  - Locate the following line:
  - Options Indexes FollowSymLinks
  - ...and append "ExecCGI":
  - Options Indexes FollowSymLinks ExecCGI
  - ...this tells Apache that CGI/Perl scripts are allowed outside of the cgi-bin directory

How to install & configure Apache on a Windows server

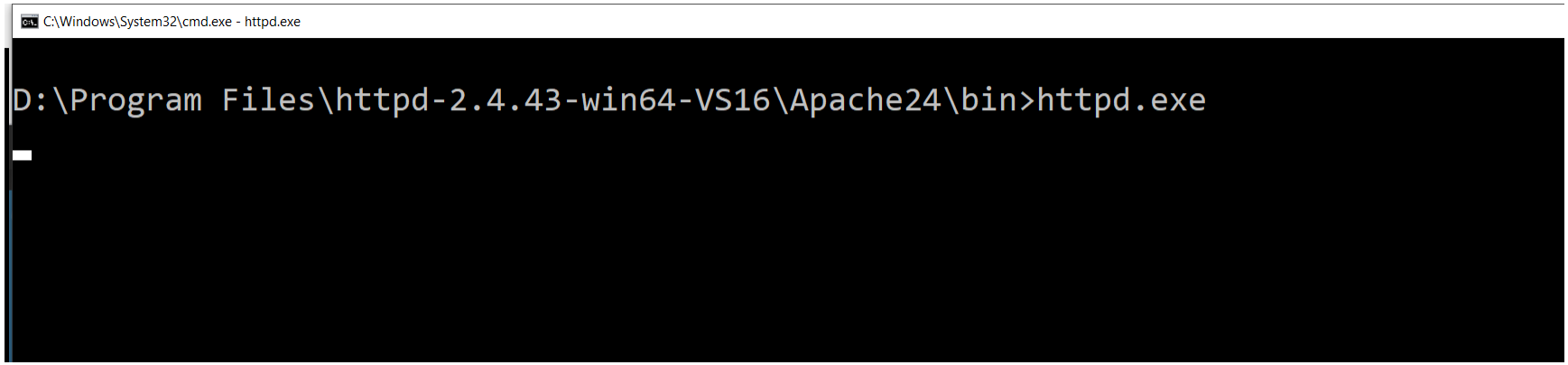
- **Step 3 - Configure Apache**
  - B) Locate and uncomment the following line: (by removing the # symbol from the start of the line)
  - AddHandler cgi-script .cgi
  - ...and also add the following line:
  - AddHandler cgi-script .pl
  - ...These two lines tell Apache how to handle .cgi/.pl files (i.e. execute them rather than present them to as text to a web browser)

How to install & configure Apache on a Windows server

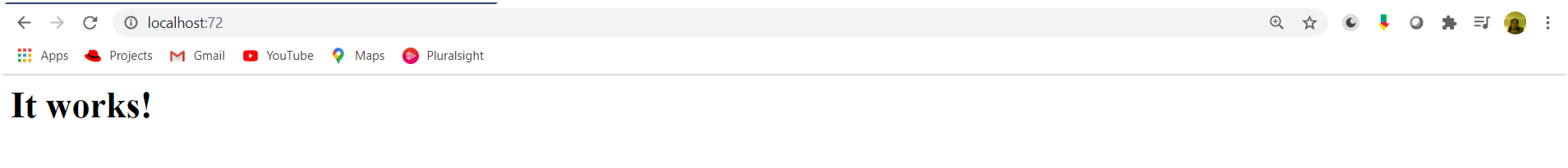
- **Step 3 - Configure Apache**
  - C) Add the following line to the end of the httpd.conf file:
  - ScriptInterpreterSource Registry
  - ...this allows Apache to ignore the very first line of .cgi/.pl files which direct Apache to the install location of Perl, and instead determine the location of Perl from the Windows Registry

How to install & configure Apache on a Windows server

- **Step 4 - Start Apache**
- Open an command/PowerShell prompt in the "bin" folder at the location where you extracted Apache (Hold "Shift" whilst right-clicking and select "Open command window here" or "Open PowerShell window here"):

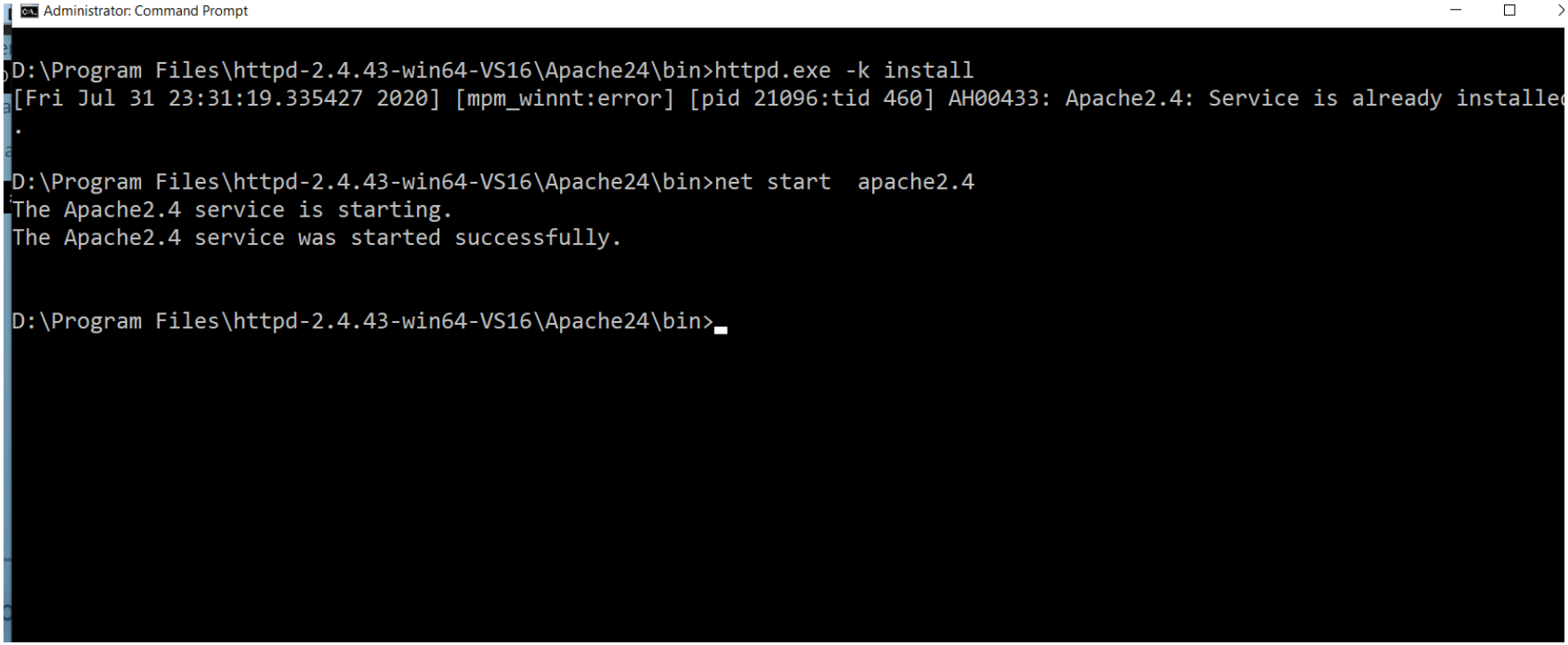


How to install & configure Apache on a Windows server



How to install & configure Apache on a Windows server

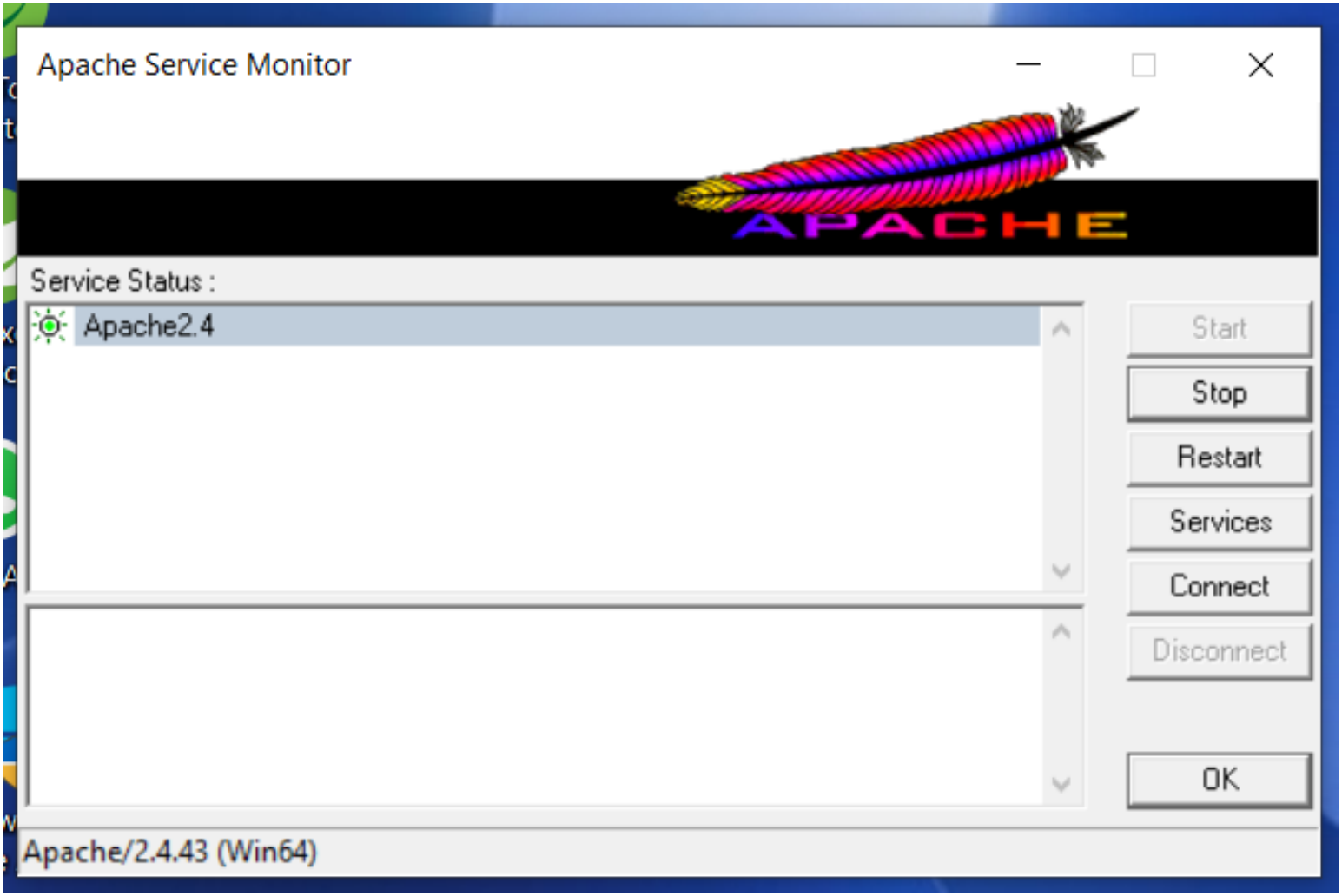
Step 6: Start as Window Service



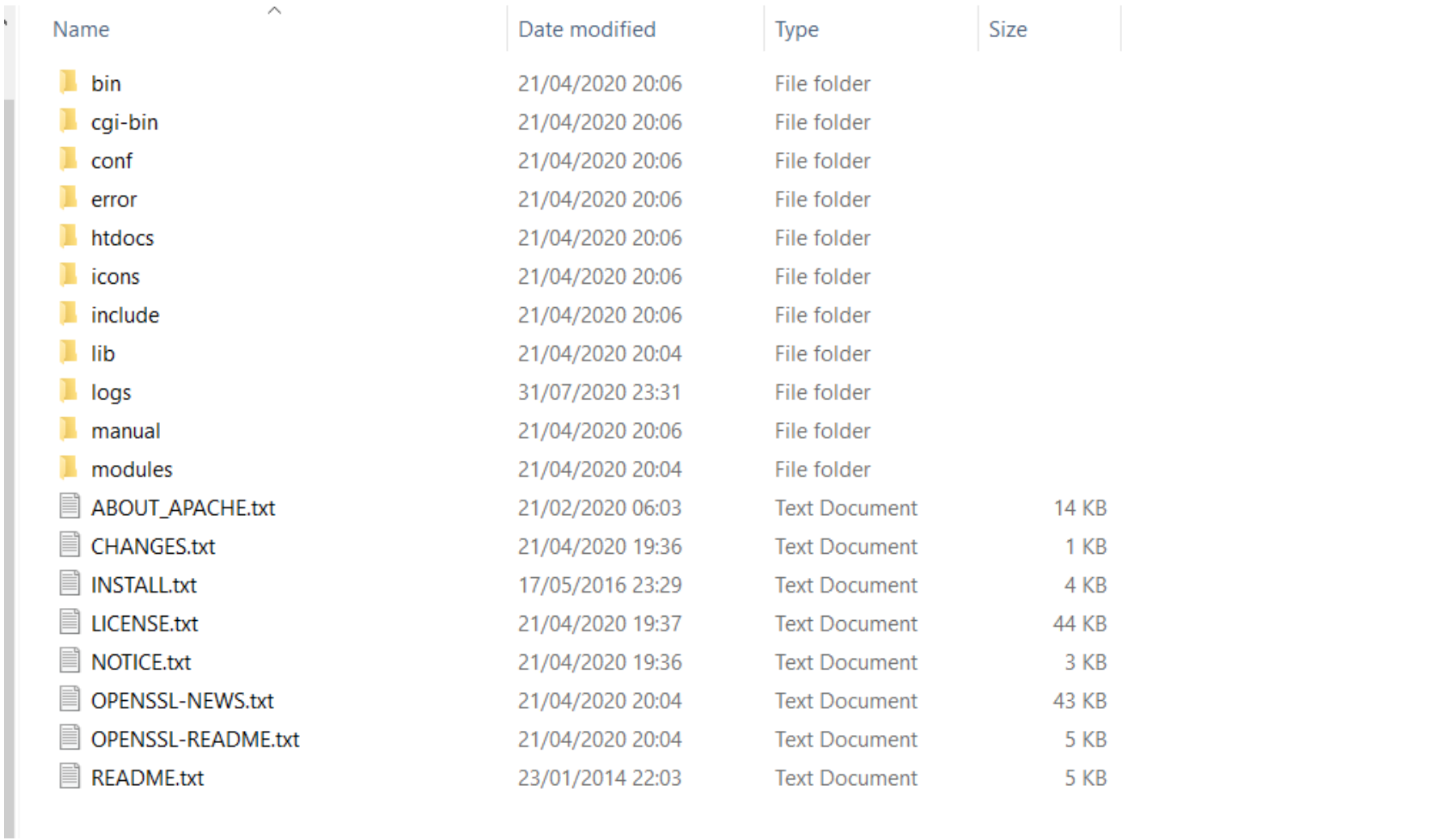
How to install & configure Apache on a Windows server

- Step 7 - Monitor Apache (optional)
- To allow you to monitor the current state of your Apache server, as well as allow you to quickly start/stop/restart the server, Apache comes with a small utility called "Apache Monitor".
- Double click ApacheMonitor.exe from the bin folder to run the utility, or place a shortcut to in in your Startup folder so that it automatically runs whenever Windows starts.

How to install & configure Apache on a Windows server



Apache Directory structure



Name	Date modified	Type	Size
bin	21/04/2020 20:06	File folder	
cgi-bin	21/04/2020 20:06	File folder	
conf	21/04/2020 20:06	File folder	
error	21/04/2020 20:06	File folder	
htdocs	21/04/2020 20:06	File folder	
icons	21/04/2020 20:06	File folder	
include	21/04/2020 20:06	File folder	
lib	21/04/2020 20:04	File folder	
logs	31/07/2020 23:31	File folder	
manual	21/04/2020 20:06	File folder	
modules	21/04/2020 20:04	File folder	
ABOUT_APACHE.txt	21/02/2020 06:03	Text Document	14 KB
CHANGES.txt	21/04/2020 19:36	Text Document	1 KB
INSTALL.txt	17/05/2016 23:29	Text Document	4 KB
LICENSE.txt	21/04/2020 19:37	Text Document	44 KB
NOTICE.txt	21/04/2020 19:36	Text Document	3 KB
OPENSSL-NEWS.txt	21/04/2020 20:04	Text Document	43 KB
OPENSSL-README.txt	21/04/2020 20:04	Text Document	5 KB
README.txt	23/01/2014 22:03	Text Document	5 KB

Apache Directory structure

- include—Contains all the C header (include) files that are only needed if you develop Web applications that integrate with Apache or want to use use third-party software with Apache, which might require the header files.
- On a production server you can remove this directory.

Apache Directory structure

- lib — Houses the Apache Portable Run-Time (APR) library files, the files that are required for running Apache, and other support utilities such as ab.
- bin—Contains the programs shown in Table .
- conf—Houses the configuration files for Apache. It contains the files listed in Table

Apache Directory structure

- htdocs—This is the default document root directory for the main Apache server. The httpd.conf file sets the DocumentRoot directive to this directory.
- By default the htdocs directory also has the entire Apache manual installed in a subdirectory.
- icons—Used to store various Apache icons that are needed for displaying dynamically the build directory listing.

Apache Directory structure

- logs—Used to store the Apache server logs, the CGI daemon socket (cgisock), and the PID file (httpd.pid).
- cgi-bin—The default CGI script directory, which is set by using the ScriptAlias directive in httpd.conf. By default, Apache comes with two simple CGI scripts—printenv and test-cgi. Each of these scripts prints out CGI environment variables when requested via http://server_name/cgi-bin/script_name URL. These scripts are good for testing whether CGI configuration is working for you.

Apache Directory structure

- logs—Used to store the Apache server logs, the CGI daemon socket (cgisock), and the PID file (httpd.pid).
- cgi-bin—The default CGI script directory, which is set by using the ScriptAlias directive in httpd.conf. By default, Apache comes with two simple CGI scripts—printenv and test-cgi. Each of these scripts prints out CGI environment variables when requested via http://server_name/cgi-bin/script_name URL. These scripts are good for testing whether CGI configuration is working for you.

Apache Directory structure

- It is highly recommend that you remove the printenv and test-cgi scripts after you have your CGI configuration working.
- It is not a good idea to have a script that displays your system environment information to anyone in the world.
- The less you tell the world about how your system works or about what is available on your system, the more secure your system remains.

Apache Directory structure

Apache Programs in the bin Directory

Apache Programs	*Purpose*
`ab`	This is the ApacheBench program. It enables you to benchmark Apache server.
`apachectl`	This is a handy script that enables you to start, restart, and stop Apache server.
`apxs`	This is a tool for building and installing extension modules for Apache. It allows you to build DSO modules that can be used in Apache by using the `mod_so` module. For more information on this program, see `http://your_server_name/manual/programs/apxs.htm`.
`htdigest`	This program creates and updates user authentication information when message digest (MD5) authentication is being used.

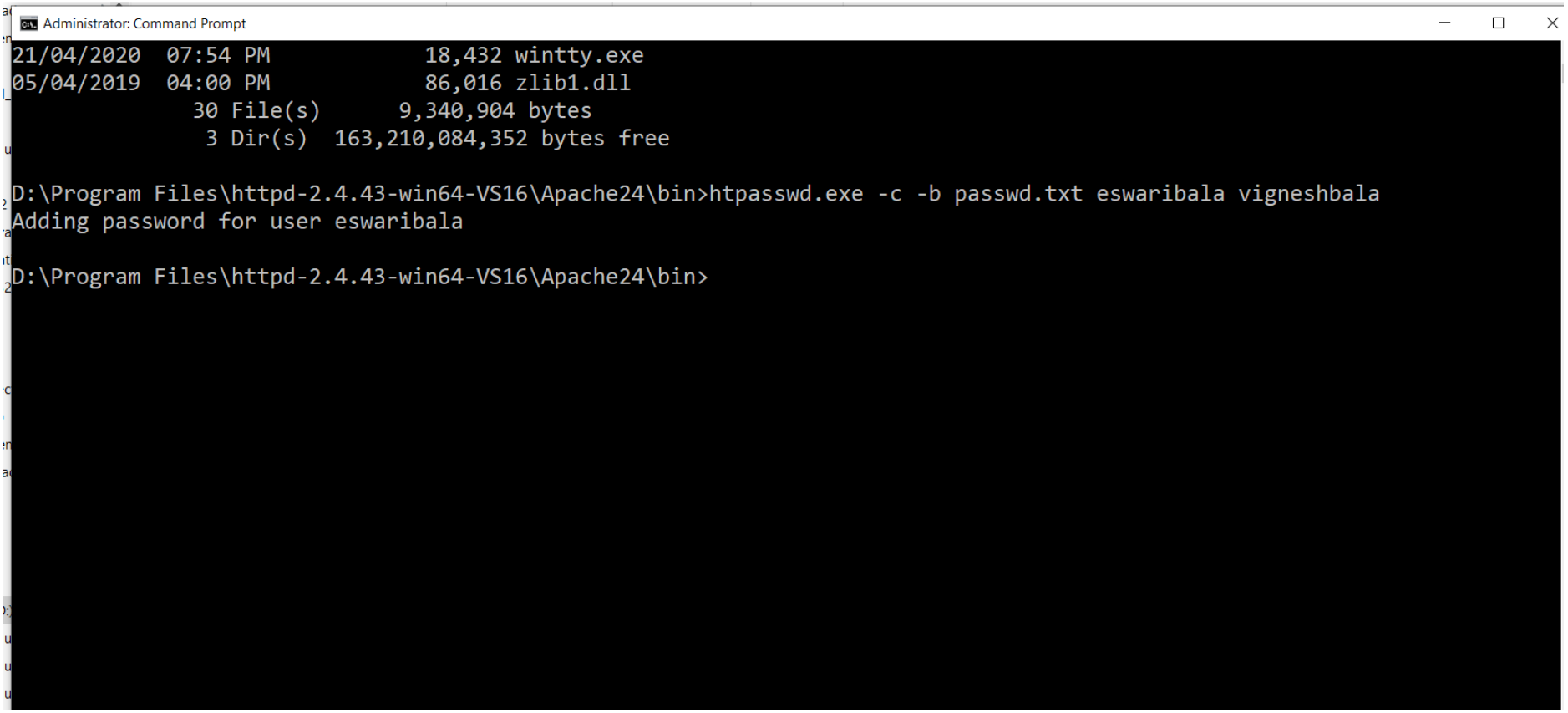
Apache Directory structure

Apache Programs	*Purpose*
`htpasswd`	This program is used to create and update user authentication information used in basic HTTP authentication.
`httpd`	This is the Apache Web server program.
`logresolve`	This program converts (resolves) IP addresses from a log file to host names.
`rotatelogs`	This program rotates Apache logs files when they reach a certain size.

Apache Directory structure

Apache config Directory Contents	
Configuration File	***Purpose***
`httpd.conf`	This is the Apache configuration file.
`httpd-std.conf`	This is the sample copy of the `httpd.conf` file, which is not required by Apache. For new Apache users, this file can act as a means for recovering the default `httpd.conf`.
`highperformance.conf`	This is a sample configuration file that shows some pointers for configuring Apache for high performance.
`highperformance-std.conf`	This is a sample copy of the `highperformance.conf` file, which is not required by Apache.
`magic`	This file stores the magic data for `mod_mime_magic` Apache module.
`mime.types`	This file is used to decide which MIME-type headers are sent to the Web client for a given file. For more information about MIME types, please read RFC 2045, 2046, 2047, 2048, and 2077. The Internet media-type registry is at `ftp://ftp.iana.org/in-notes/iana/assignments/media-types`.

Apache Directory structure



Configuring Apache

- By default, Apache reads a single configuration file called httpd.conf.
- Every Apache source distribution comes with a set of sample configuration files.
- In the standard Apache source distribution, you will find a directory called conf, which contains sample configuration files with the –dist extension.
- The very first step you need to take before you modify this file is to create a backup copy of the original.

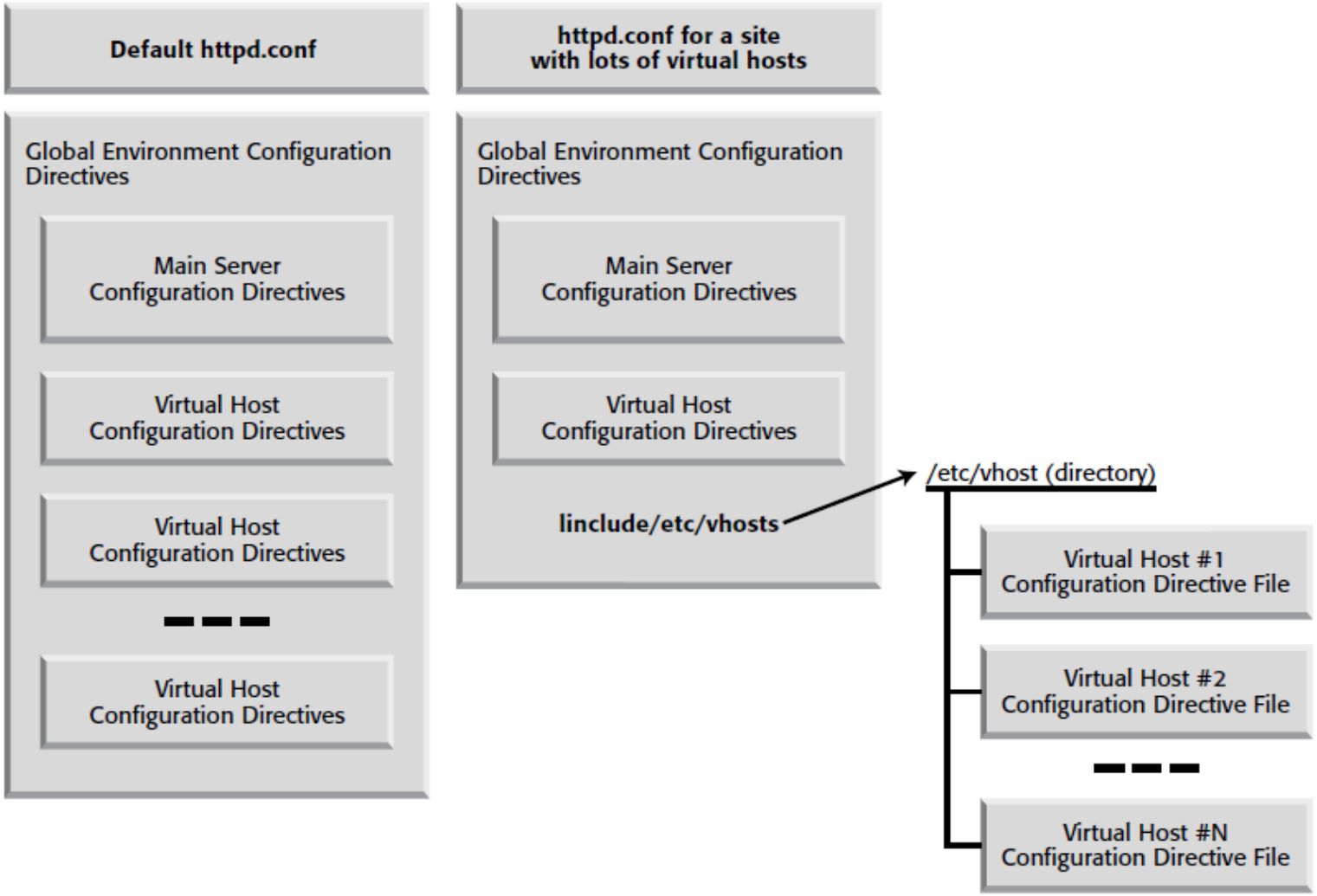
Configuring Apache

- The httpd.conf file has two types of information: comments and server directives.
- Lines starting with a leading # character are treated as a comment line; these comments have no purpose for the server software, but they serve as a form of documentation for the server administrator.
- You can add as many comments as you want; the server simply ignores all comments when it parses the file.

Configuring Apache

- Except for the comments and blank lines, the server treats all other lines as either complete or partial directives.
- A directive is like a command for the server. It tells the server to do a certain task in a particular fashion.
- While editing the httpd.conf file, you need to make certain decisions regarding how you want the server to behave.

Httpd.conf Segments



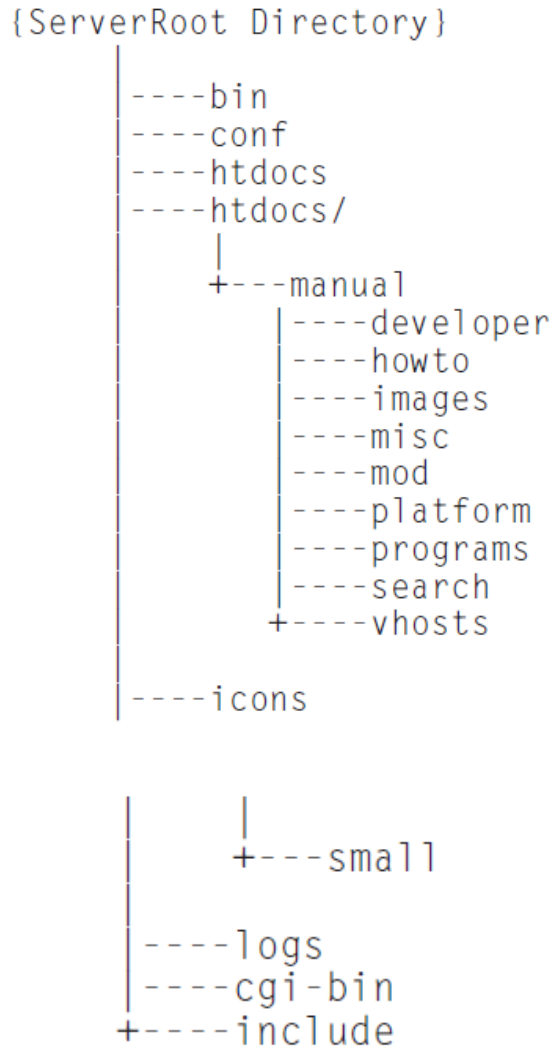
Httpd.conf Segments

- The left side of the figure shows how the default httpd.conf can be visualized in your mind.
- There are configuration directives that create the global server environment that applies to everything; there are configuration options that apply to the main (default) Web site Apache servers, and there are configuration directives that only apply to optional virtual hosts.
- Because Apache uses a single configuration file, a site with lots of virtual hosts will have a very large file and management of the configuration becomes very cumbersome.

Configuring the global environment for Apache

- The very first directive is ServerRoot, which appears as follows:
- ServerRoot “/usr/local/apache”
- This directive specifies the top-level directory of the Web server.
- The specified directory is not where you keep your Web contents.
- It is really a directory, which normally has these subdirectories:

Configuring the global environment for Apache



Configuring the global environment for Apache

- /usr/local/apache is the parent directory for all server-related files.
- The default value for ServerRoot is set to whatever you choose for --prefix option during source configuration using the configure script.
- By default, the make install command executed during server installation copies all the server binaries in %ServerRoot%/bin, server configuration files in %ServerRoot%/conf, and so on

Configuring the global environment for Apache

- You should only change the value of this directive if you have manually moved the entire directory from the place on installation to another location.
- For example, if you simply run cp -r /usr/local/apache /home/apache and want to configure the Apache server to work from the new location, you will change this directive to ServerRoot /home/apache.
- Note that in such a case you will also have to change other direct references to /usr/local/apache to /home/apache.

Configuring the global environment for Apache

- Also note that whenever you see a relative directory name in the configuration file, Apache will prefix %ServerRoot% to the path to construct the actual path.

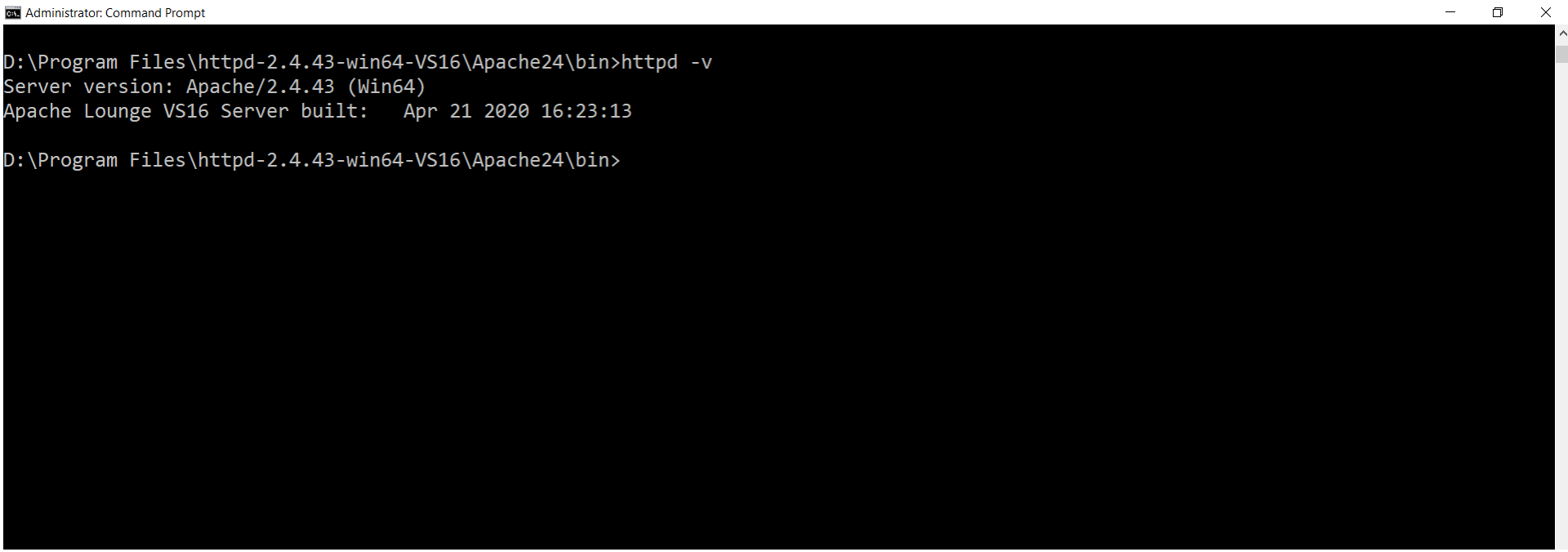
PidFile

- The PidFile directive sets the PID (process ID) file path.
- By default, it is set to logs/httpd.pid, which translates to %ServerRoot%/logs/httpd.pid (that is, /usr/local/apache/logs/httpd.pid).
- Whenever you want to find the PID of the main Apache process that runs as root and spawns child processes, you can run the cat %ServerRoot/logs/httpd.pid command.
- Don’t forget to replace %ServerRoot% with an appropriate value.

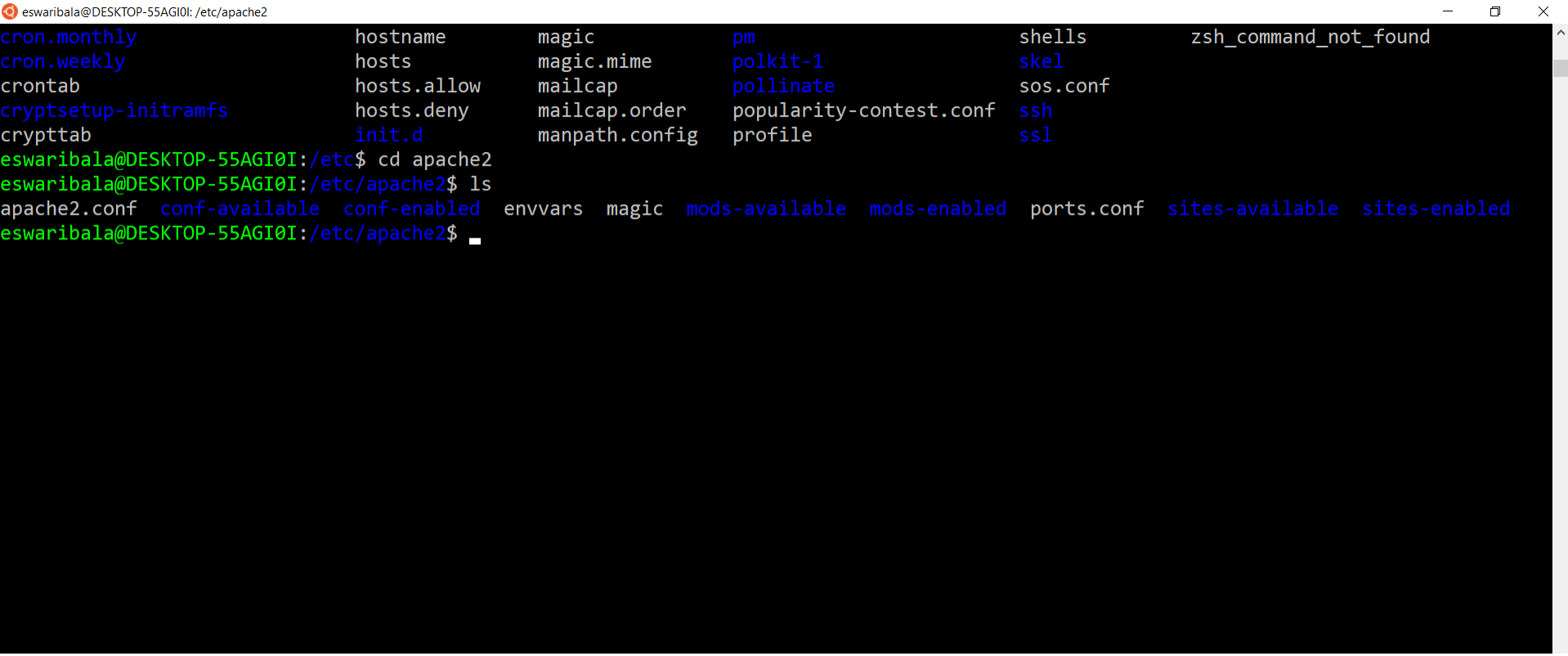
eswaribala@DESKTOP-55AGI0I: /

```
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ cd eenvvars
-bash: cd: eenvvars: No such file or directory
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ cd envvars
-bash: cd: envvars: Not a directory
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ vi envvars
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ vi apache2.conf
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ cat %ServerRoot/logs/apache2.pid
cat: %ServerRoot/logs/apache2.pid: No such file or directory
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ ls
apache2.conf  conf-available  conf-enabled  envvars  magic  mods-available  mods-enabled  ports.conf  sites-available  sites-enabled
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ vi apache2.conf
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ sudo nano envvars
[sudo] password for eswaribala:
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ cd ..
eswaribala@DESKTOP-55AGI0I:/etc$ cd ..
eswaribala@DESKTOP-55AGI0I:/$ ls var
backups  cache  crash  lib  local  lock  log  mail  opt  run  snap  spool  tmp  www
eswaribala@DESKTOP-55AGI0I:/$ ls var/run/apache2
apache2.pid
eswaribala@DESKTOP-55AGI0I:/$ vi  var/run/apache2/apache2.pid
eswaribala@DESKTOP-55AGI0I:/$ ps
  PID TTY          TIME CMD
   10 tty1     00:00:00 bash
  675 tty1     00:00:00 ps
eswaribala@DESKTOP-55AGI0I:/$ vi  var/run/apache2/apache2.pid
eswaribala@DESKTOP-55AGI0I:/$ ps
  PID TTY          TIME CMD
   10 tty1     00:00:00 bash
  677 tty1     00:00:00 ps
eswaribala@DESKTOP-55AGI0I:/$
```

PidFile


Administrator: Command Prompt
D:\Program Files\httpd-2.4.43-win64-VS16\Apache24\bin>httpd -v
Server version: Apache/2.4.43 (Win64)
Apache Lounge VS16 Server built: Apr 21 2020 16:23:13
D:\Program Files\httpd-2.4.43-win64-VS16\Apache24\bin>

Pid file



D:\Program Files\httpd-2.4.43-win64-VS16\Apache24\logs

ScoreBoardFile

ScoreBoardFile is encapsulated within an if condition by using the

<IfModule . . .> container as shown below:

<IfModule !perchild.c>

ScoreBoardFile logs/apache_runtime_status

</IfModule>

– This tells Apache to set the ScoreBoardFile to %ServerRoot%/logs/ apache_runtime_status file only if you have chosen a multiprocessing module (MPM) other than perchild.

ScoreBoardFile

Because the default MPincluding Linux, is threaded instead of perchild, the if condition will be true and Apache will set the ScoreBoardFile directive.

This directive is used to point to a file, which is used to exchange run-time status information between Apache processesM for most operating systems,

Timeout, KeepAlive, MaxKeepAliveRequests, and KeepAliveTimeout

- Timeout sets the server timeout in seconds. The default should be left alone.
- The next three directives KeepAlive, MaxKeepAliveRequests, and KeepAliveTimeout are used to control the keep-alive behavior of the server.
- You do not need to change them.

Timeout, KeepAlive, MaxKeepAliveRequests, and KeepAliveTimeout

- IfModule containers Apache will use one of the next three <IfModule . . .> containers based on which MPM you chose.
- For example, if you configured Apache using the default MPM mode (threaded) on a Linux system, then the following <IfModule . . .> container will be used:
- <IfModule threaded.c>
- StartServers 3
- MaxClients 8
- MinSpareThreads 5
- MaxSpareThreads 10
- ThreadsPerChild 25
- MaxRequestsPerChild 0
- </IfModule>

Timeout, KeepAlive, MaxKeepAliveRequests, and KeepAliveTimeout

- On the other hand, if you chose --with-mpm=prefork during source configuration by using the configure script, then the following <IfModule . . .> container will be used:
- <IfModule prefork.c>
- StartServers 5
- MinSpareServers 5
- MaxSpareServers 10
- MaxClients 20
- MaxRequestsPerChild 0
- </IfModule>
- Similarly, the --with-mpm=perchild option forces Apache to use the last <IfModule . . .> container

Directives for threaded (default) MPM behavior

- If you did not change the default MPM behavior during source compilation and used the threaded behavior, so the directives that you need to consider are
- StartServers
- MaxClients
- MinSpareThreads
- MaxSpareThreads
- ThreadsPerChild
- MaxRequestPerChild

StartServers

- StartServers tells Apache to start three child servers as it starts.
- You can start more servers if you want, but Apache is pretty good at increasing number of child processes as needed based on load.
- So, changing this directive is not required.

MaxClients

- In the default threaded MPM mode, the total number of simultaneous requests that Apache can process is %MaxClients% x %ThreadsPerChild%.
- So, because the default for MaxClients is 8 and the default for ThreadsPerChild is 25, the default maximum for simultaneous requests is 200 (that is, 8 times 5).
- If you use the preforking MPM mode, the maximum requests is limited to %MaxClients%.
- The default maximum of 200 simultaneous requests should work well for most sites, so leave the defaults.

MinSpareThreads

- The MinSpareThreads directive specifies the minimum number of idle threads.
- These spare threads are used to service requests and new spare threads are created to maintain the minimum spare thread pool size.
- You can leave the default settings alone.

MaxSpareThreads

- The MaxSpareThreads directive specifies the maximum number of idle threads;leave the default as is.
- In the default threaded mode, Apache kills child processes to control minimum and maximum thread count.

ThreadsPerChild

- This directive defines how many threads are created per child process.
- If you are running Apache on a Windows system, set ThreadsPerChild to the maximum number of simultaneous requests that you want to handle, because on this platform there is only one child process, and it owns all the threads.

MaxRequestPerChild

- The final directive for the global environment is MaxRequestPerChild, which sets the number of requests a child process can serve before getting killed.
- The default value of zero makes the child process serve requests forever.
- I do not like to the default value because it enables Apache processes to slowly consume large amounts of memory when a faulty mod_perl script, or even a faulty third-party Apache module, leaks memory.

MaxRequestPerChild

- If you do not plan on running any third-party Apache modules or mod_perl scripts, you can keep the defaults or else set it to a reasonable number.
- A setting of 30 ensures that the child process is killed after processing 30 requests.

Configuring the main server

- The main server configuration applies to the default Web site Apache serves.
- This is the site that will come up when you run Apache and use the server's IP address or host name on a Web browser.

Port

- The very first directive in this section is the Port directive, which sets the TCP port that Apache listens to for connections.
- The default value of 80 is the standard HTTP port.
- If you change this to another number, such as 8080, you can only access the server using a URL such as http://hostname:8080/.
- You must specify the port number in the URL if the server runs on a nonstandard port.

Port

- There are many reasons for running Apache on nonstandard ports, but the only good one I can think of is that you do not have permission to run Apache on the standard HTTP.

User and Group directives

- The User and Group directives tell Apache the user (UID) and group (GID) names to use.
- These two directives are very important for security reasons.
- When the primary Web server process launches a child server process to fulfill a request, it changes the child's UID and GID according to the values set for these directives.
- Refer to Figure to see how the primary Web server process that listens for the connection runs as a root user process, and how the child processes run as different user/group processes.
- If the child processes are run as root user processes, a potential security hole will be opened for attack by hackers.

User and Group directives

- Enabling the capability to interact with a root user process maximizes a potential breach of security in the system; hence, this is not recommended.
- Rather, I highly recommend that you choose to run the child server processes as a very low-privileged user belonging to a very low-privileged group.
- In most Unix systems, the user named nobody (usually UID = -1) and the group named nogroup (usually GID = -1) are low-privileged.
- You should consult your /etc/group and /etc/passwd files to determine these settings.

User and Group directives

- If you plan to run the primary Web server as a nonroot (regular) user, it will not be able to change the UID and GID of child processes, because only root user processes can change the UID or GID of other processes.
- Therefore, if you run your primary server as the user named ironsheik, then all child processes will have the same privileges as ironsheik.
- Similarly, whatever group ID you have will also be the group ID for the child processes.

ServerAdmin

- ServerAdmin defines the e-mail address that is shown when the server generates an error page.
- Set this to your e-mail address.ServerName.
- Now you need to set the host name for the Server using the ServerName directive.
- This directive is commented out by default because Apache install cannot guess what host name to use for your system.
- So if the host name is called www.domain.com, set ServerName directive accordingly.

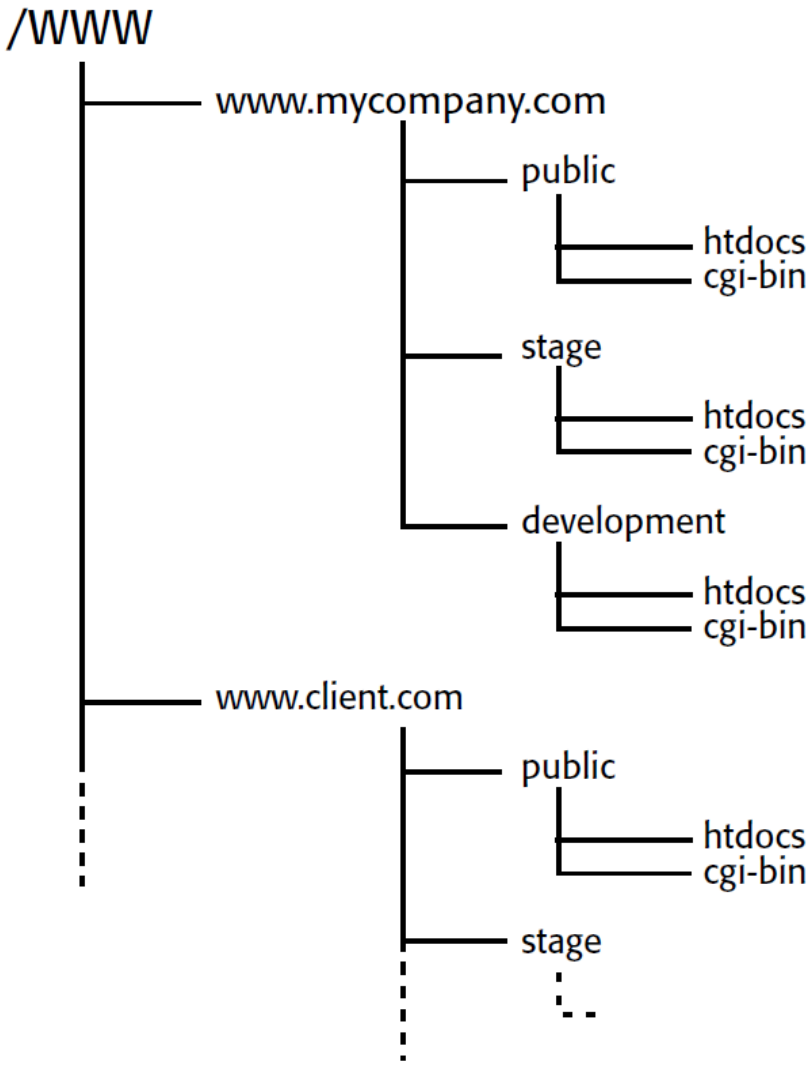
DocumentRoot

- Like all other Web servers, Apache needs to know the path of the top-level directory where Web pages will be kept.
- This directory is typically called the document root directory. Apache provides a directive called DocumentRoot, which can be used to specify the path of the top-level Web directory.
- This directive instructs the server to treat the supplied directory as the root directory for all documents. This is a very important decision for you to make.
- For example, if the directive is set as:
- DocumentRoot / then every file on the system becomes accessible by the Web server

DocumentRoot

- You can protect files by providing proper file permission settings, but setting the document root to the physical root directory of your system is definitely a major security risk.
- Instead, you should point the document root to a specific subdirectory of your file system.
- If you have used the --prefix=/usr/local/apache option in configuring the Apache source, this directive will be set as:
- DocumentRoot “/usr/local/apache/htdocs”

Preferred Web Directory Structure



Preferred Web Directory Structure

- I chose to create a partition called /www, and under it there are subdirectories for each Web site hosted by my system.
- /www/www.mycompany.com/ has three subdirectories:
- public, stage, and development.
- Each of these subdirectories has two subdirectories: htdocs and cgi-bin.
- The htdocs subdirectory is the document root directory, and the cgi-bin subdirectory is used for CGI scripts.
- So, the DocumentRoot setting for the www.mycompany.com Web site is:
- DocumentRoot “/www/www.mycompany.com/public/htdocs”

Preferred Web Directory Structure

- The advantage of this directory structure is that it keeps all Web documents and applications under one partition (/www).
- This enables easy backups, and the partition can be mounted on different systems via the Network File System (NFS) in case another machine in the network is given the task to provide Web services.

Directory container directives

- The next set of directives are enclosed in a <Directory . . .> container as
- shown here:
- <Directory />
- Options FollowSymLinks
- AllowOverride None
- </Directory>
- The scope of the enclosed directives is limited to the named directory (with any subdirectories); however, you may only use directives that are allowed in a directory context.

Directory container directives

- Here the Options and the AllowOverride directives apply to %DocumentRoot% that is root (/) or the top-level directory of the main Web site.
- Because directives enclosed within a directory container apply to all the subdirectories below the named directory, the directives apply to all directories within %DocumentRoot%.

Directory container directives

- The Options directive is set to FollowSymLinks, which tells Apache to allow itself to traverse any symbolic within %DocumentRoot%.
- Because the Options directive is only set to follow symbolic links, no other options are available to any of the directories within %DocumentRoot%.
- Effectively, the Options directive is:
- Options FollowSymLinks -ExecCGI -Includes -Indexes -MultiViews

Apache Rocks On

- Apache runs on Linux and other Unix systems.
- Apache also runs on Windows.
- Installing Apache
- To install Apache, install the latest meta-package apache2 by running:
  - sudo apt update
  - sudo apt install apache2
  - Adjusting Firewall

    sudo ufw app list

    sudo ufw allow 'apache'

Apache Rocks On

- Apache runs on Linux and other Unix systems.
- Apache also runs on Windows.
- Installing Apache
- To install Apache, install the latest meta-package apache2 by running:
  - sudo apt update
  - sudo apt install apache2
  - Adjusting Firewall

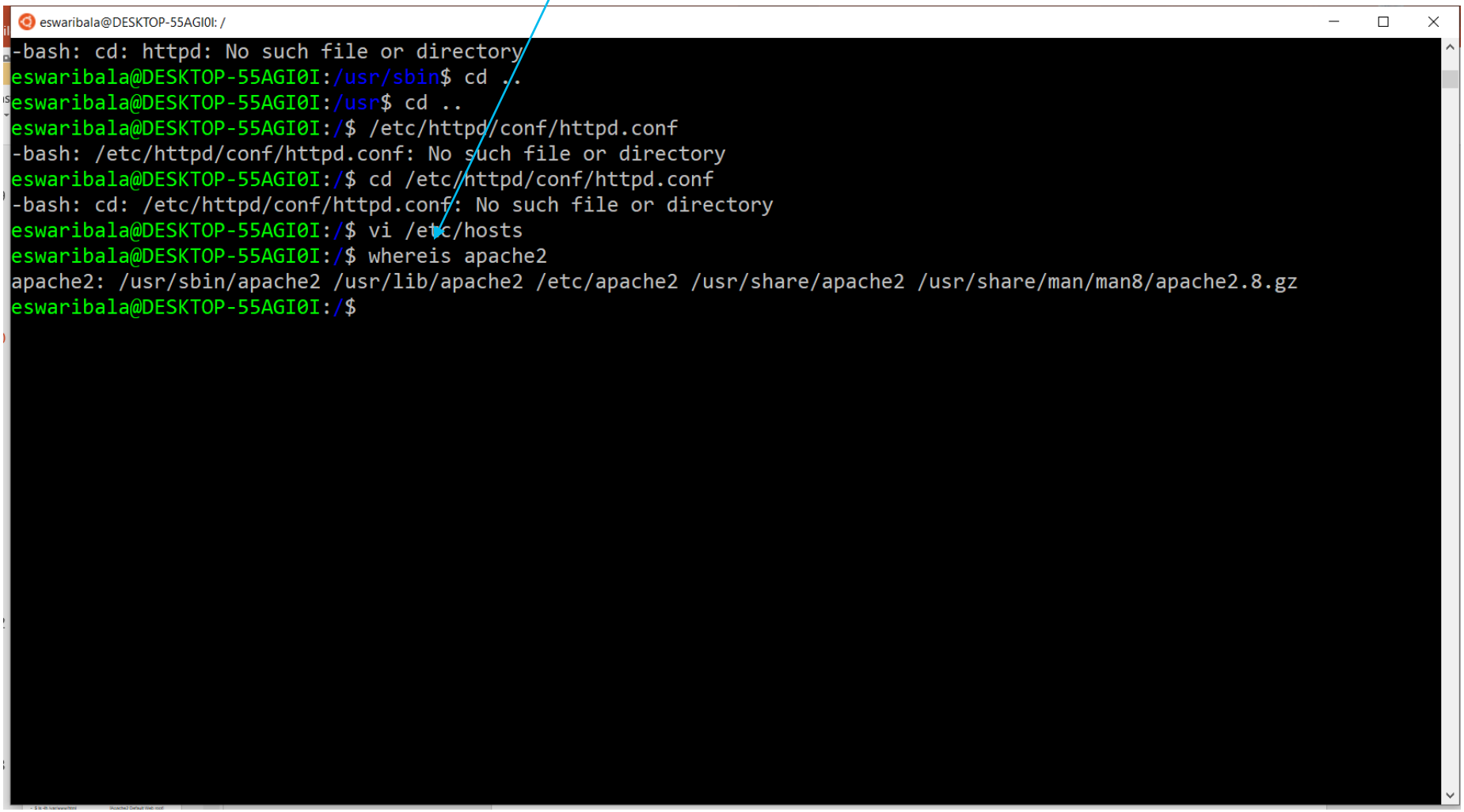
    sudo ufw app list

    sudo ufw allow 'apache'

Apache Rocks On

- sudo ufw status
- Sudo whereis apache2
- sudo service apache2 start
- sudo service --status-all
- sudo apache2ctl start
- sudo vi /etc/apache2/ports.conf //to change port number
- #to suppress warning when we start server
- #add this in ports.conf
- **AcceptFilter https none**
- **AcceptFilter http none**

Apache Rocks On



Apache Rocks On

- sudo service apache2 stop
- sudo service apache2 restart
- sudo service apache2 reload (after config changes)
- sudo apachectl -t (test config changes)
- sudo apachectl configtest
- sudo apache2 –v
- Sudo apache2ctl –S (to find errors in conf)

localhost:76

Apps Projects Gmail YouTube Maps Pluralsight

Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

List Important configuration files

- $ ls -lh /usr/lib/apache2/modules/ [To List Available Modules]
- $ ls -lh /etc/apache2/mods-available/ [To List Available Modules]
- $ ls -lh /etc/apache2/mods-enabled/ [To List Enabled Modules]
- $ ls -lh /var/www/html [Apache2 Default Web root]
- $ ls -lh /var/www/ [Other Website Web root]
- $ less /etc/apache2/apache2.conf [Apache2 Main Configuration File]
- $ less /etc/apache2/ports.conf [Apache2 Ports Configuration File]
- $ less /var/log/apache2/error.log [Apache2 Error Log File]
- $ less /var/log/apache2/access.log [Apache2 Access Log File]

Modify Default Sites

- $ sudo mkdir -p /var/www/test.com
- $ cd /var/www/test.com
- $ sudo nano index.html
- <head>
- <title>Test.com index page</title>
- </head>
- <h1>Hello, welcome to test.com! It works!</h1>
- <h2>That is all I have to say. If you don't see this then it doesn't work.</h2>
- </body>
- </html>

Modify Default Sites

- $ cd /etc/apache2/sites-available/
- $ sudo nano test.com.conf
- <VirtualHost *:80>
- ServerAdmin carla@localhost
- DocumentRoot /var/www/test.com
- ServerName test.com
- ServerAlias www.test.com
- ErrorLog ${APACHE_LOG_DIR}/error.log
- CustomLog ${APACHE_LOG_DIR}/access.log combined
- </VirtualHost>

Modify Default Sites

- $ sudo a2ensite test.com.conf
- Enabling site test.com.
- To activate the new configuration, you need to run:
- service apache2 reload

Configuration

- Apache2 is configured by placing directives in plain text configuration files.
- These directives are separated between the following files and directories:
- **/etc/apache2**: The Apache configuration directory.
  - All of the Apache configuration files reside here.
- **/etc/apache2/apache2.conf**: The main Apache configuration file.
  - This can be modified to make changes to the Apache global configuration.
  - This file is responsible for loading many of the other files in the configuration directory.

Configuration

- **httpd.conf**: historically the main Apache2 configuration file, named after the httpd daemon**. In other distributions (or older versions of Ubuntu), the file might be present**. In Ubuntu, **all configuration options have been moved to apache2.conf** and the below referenced directories, and this file no longer exists.

Configuration

- **/etc/apache2/ports.conf**: This file specifies the ports that Apache will listen on.
  - By default, Apache listens on port 80 and additionally listens on port 443 when a module providing SSL capabilities is enabled.
- **/etc/apache2/sites-available/:** The directory where per-site virtual hosts can be stored.
  - Apache will not use the configuration files found in this directory unless they are linked to the sites-enabled directory.
  - Typically, all server block configuration is done in this directory, and then enabled by linking to the other directory with the a2ensite command.

Configuration

- **/etc/apache2/sites-enabled/:** The directory where enabled per-site virtual hosts are stored.
  - Typically, these are created by linking to configuration files found in the sites-available directory with the a2ensite.
  - Apache reads the configuration files and links found in this directory when it starts or reloads to compile a complete configuration.
- **/etc/apache2/conf-available/, /etc/apache2/conf-enabled/:** These directories have the same relationship as the sites-available and sites-enabled directories, but are used to store configuration fragments that do not belong in a virtual host.
  - Files in the conf-available directory can be enabled with the a2enconf command and disabled with the a2disconf command.

Configuration

- **conf-available**: this directory contains available configuration files. All files that were previously in /etc/apache2/conf.d should be moved to /etc/apache2/conf-available.
- **conf-enabled:** holds symlinks to the files in /etc/apache2/conf-available. When a configuration file is symlinked, it will be enabled the next time apache2 is restarted.
- **envvars**: file where Apache2 environment variables are set.

Configuration

- **/etc/apache2/mods-available/, /etc/apache2/mods-enabled/:** These directories contain the available and enabled modules, respectively.
  - Files in ending in .load contain fragments to load specific modules, while files ending in .conf contain the configuration for those modules.
  - Modules can be enabled and disabled using the a2enmod and a2dismod command.

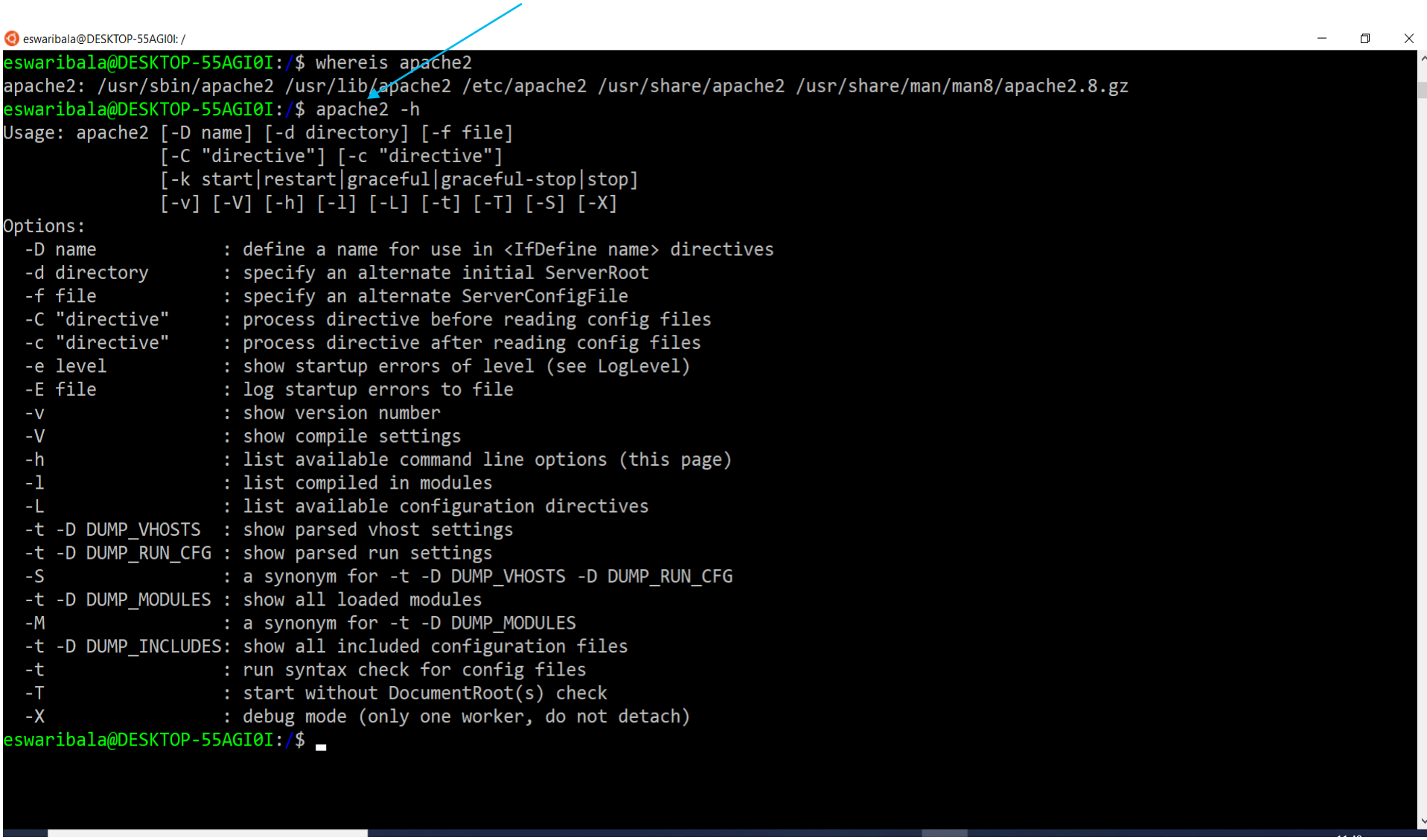
Configuration

- **mods-enabled:** holds symlinks to the files in /etc/apache2/mods-available. When a module configuration file is symlinked it will be enabled the next time apache2 is restarted.
- **ports.conf**: houses the directives that determine which TCP ports Apache2 is listening on.
- **magic**: instructions for determining MIME type based on the first few bytes of a file.

Server Logs

- **/var/log/apache2/access.log**: By default, every request to your web server is recorded in this log file unless Apache is configured to do otherwise.
- **/var/log/apache2/error.log**: By default, all errors are recorded in this file. The LogLevel directive in the Apache configuration specifies how much detail the error logs will contain.

Apache2 Options



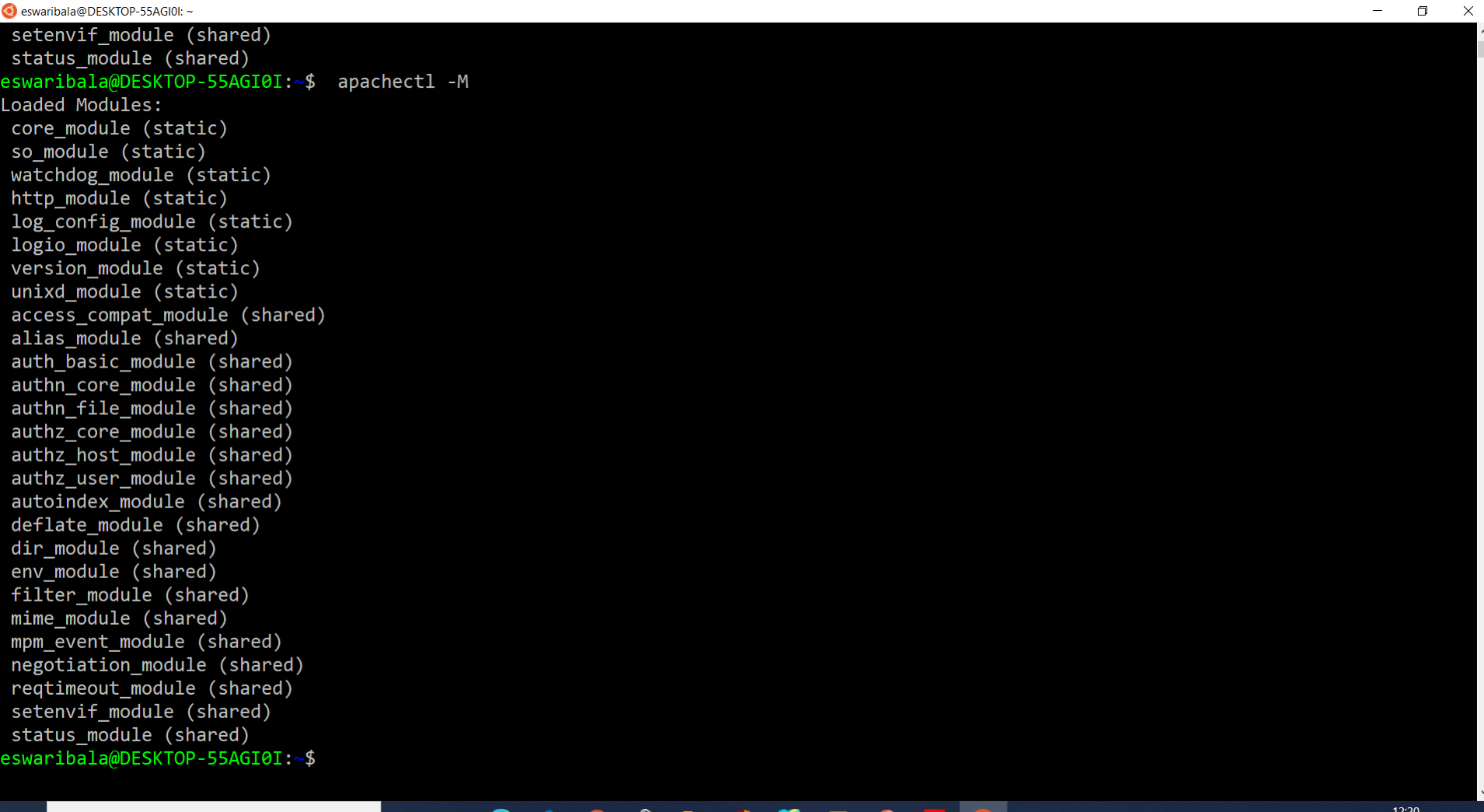
Show All Loaded Modules in Apache 2

Apache2ctl –t -D DUMP_MODULES

```
eswaribala@DESKTOP-55AGI0I:~$ apache2ctl -t -D DUMP_MODULES
Loaded Modules:
 core_module (static)
 so_module (static)
 watchdog_module (static)
 http_module (static)
 log_config_module (static)
 logio_module (static)
 version_module (static)
 unixd_module (static)
 access_compat_module (shared)
 alias_module (shared)
 auth_basic_module (shared)
 authn_core_module (shared)
 authn_file_module (shared)
 authz_core_module (shared)
 authz_host_module (shared)
 authz_user_module (shared)
 autoindex_module (shared)
 deflate_module (shared)
 dir_module (shared)
 env_module (shared)
 filter_module (shared)
 mime_module (shared)
 mpm_event_module (shared)
 negotiation_module (shared)
 reqtimeout_module (shared)
 setenvif_module (shared)
 status_module (shared)
eswaribala@DESKTOP-55AGI0I:~$
```

Show All Loaded Modules in Apache 2

Apache2ctl –M



Show list of compiled modules


```
eswaribala@DESKTOP-55AGI0I: ~
eswaribala@DESKTOP-55AGI0I:~$ apachectl -l
Compiled in modules:
  core.c
  mod_so.c
  mod_watchdog.c
  http_core.c
  mod_log_config.c
  mod_logio.c
  mod_version.c
  mod_unixd.c
eswaribala@DESKTOP-55AGI0I:~$
```

Go to Modules Folder


```
eswaribala@DESKTOP-55AGI0I:/usr/lib/apache2/modules$ cd /usr/lib/apache2/modules
eswaribala@DESKTOP-55AGI0I:/usr/lib/apache2/modules$ ls
httpd.exp               mod_autoindex.so       mod_expires.so             mod_mpm_event.so       mod_rewrite.so
mod_access_compat.so    mod_brotli.so          mod_ext_filter.so          mod_mpm_prefork.so     mod_sed.so
mod_actions.so          mod_bucketeer.so       mod_file_cache.so          mod_mpm_worker.so      mod_session.so
mod_alias.so            mod_buffer.so          mod_filter.so              mod_negotiation.so     mod_session_cookie.so
mod_allowmethods.so     mod_cache.so           mod_headers.so             mod_proxy.so           mod_session_crypto.so
mod_asis.so             mod_cache_disk.so      mod_heartbeat.so           mod_proxy_ajp.so       mod_session_dbd.so
mod_auth_basic.so       mod_cache_socache.so   mod_heartmonitor.so        mod_proxy_balancer.so  mod_setenvif.so
mod_auth_digest.so      mod_case_filter.so     mod_http2.so               mod_proxy_connect.so   mod_slotmem_plain.so
mod_auth_form.so        mod_case_filter_in.so  mod_ident.so               mod_proxy_express.so   mod_slotmem_shm.so
mod_authn_anon.so       mod_cern_meta.so       mod_imagemap.so            mod_proxy_fcgi.so      mod_socache_dbm.so
mod_authn_core.so       mod_cgi.so             mod_include.so             mod_proxy_fdpass.so    mod_socache_memcache.so
mod_authn_dbd.so        mod_cgid.so            mod_info.so                mod_proxy_ftp.so       mod_socache_redis.so
mod_authn_dbm.so        mod_charset_lite.so    mod_lbmethod_bybusyness.so mod_proxy_hcheck.so    mod_socache_shmcb.so
mod_authn_file.so       mod_data.so            mod_lbmethod_byrequests.so mod_proxy_html.so      mod_speling.so
mod_authn_socache.so    mod_dav.so             mod_lbmethod_bytraffic.so  mod_proxy_http.so      mod_ssl.so
mod_authnz_fcgi.so      mod_dav_fs.so          mod_lbmethod_heartbeat.so  mod_proxy_http2.so     mod_status.so
mod_authnz_ldap.so      mod_dav_lock.so        mod_ldap.so                mod_proxy_scgi.so      mod_substitute.so
mod_authz_core.so       mod_dbd.so             mod_log_debug.so           mod_proxy_uwsgi.so     mod_suexec.so
mod_authz_dbd.so        mod_deflate.so         mod_log_forensic.so        mod_proxy_wstunnel.so  mod_unique_id.so
mod_authz_dbm.so        mod_dialup.so          mod_lua.so                 mod_ratelimit.so       mod_userdir.so
mod_authz_groupfile.so  mod_dir.so             mod_macro.so               mod_reflector.so       mod_usertrack.so
mod_authz_host.so       mod_dumpio.so          mod_md.so                  mod_remoteip.so        mod_vhost_alias.so
mod_authz_owner.so      mod_echo.so            mod_mime.so                mod_reqtimeout.so      mod_xml2enc.so
mod_authz_user.so       mod_env.so             mod_mime_magic.so          mod_request.so
eswaribala@DESKTOP-55AGI0I:/usr/lib/apache2/modules$
```

Basic Settings

- Apache2 ships with a virtual-host-friendly default configuration.
- That is, it is configured with a single default virtual host (using the Virtual Host directive) which can be modified or used as-is if you have a single site, or used as a template for additional virtual hosts if you have multiple sites.
- If left alone, the default virtual host will serve as your default site, or the site users will see if the URL they enter does not match the ServerName directive of any of your custom sites.
- To modify the default virtual host, edit the file /etc/apache2/sites-available/000-default.conf.

Method-1: How to find the Process ID (PID) of a program running on Linux using the pidof Command

- The pidof command is used to find the process ID of the running program.
- It prints those IDs into the standard output.
- To demonstrate this, we are going to find the Apache2 process id from Ubuntu 20 system.

```
eswaribala@DESKTOP-55AGI0I: ~
eswaribala@DESKTOP-55AGI0I:~$ pidof apache2
108 107 105
eswaribala@DESKTOP-55AGI0I:~$
```

Method-1: How to find the Process ID (PID) of a program running on Linux using the pidof Command

- In the above output you may have difficulties identifying the process ID because it displays all PIDs (including parent and child) against the process name.
- So we need to find the Parent Process PID (PPID), which is what we are looking for.
- This may be the first number. In my case it is 108 and it is sorted in descending order.

Method-2: How to find the Process ID (PID) of a program running on Linux using the pgrep Command

- The pgrep command looks at the processes currently running and lists the process IDs that match the selection criteria.

Method-3: How to find the Process ID (PID) of a program running on Linux using the pstree Command

- The pstree command shows running processes as a tree.
- The tree is rooted at either pid or init if pid is omitted.
- If a user name is specified in the pstree command then it’s shows all the process owned by the corresponding user.
- pstree visually merges identical branches by putting them in square brackets and prefixing them with the repetition count.

Method-3: How to find the Process ID (PID) of a program running on Linux using the pstree Command

- The pstree command shows running processes as a tree.
- The tree is rooted at either pid or init if pid is omitted.
- If a user name is specified in the pstree command then it’s shows all the process owned by the corresponding user.
- pstree visually merges identical branches by putting them in square brackets and prefixing them with the repetition count.

To get all the process
pstree -p | grep "apache2"


```
eswaribala@DESKTOP-55AGI0I: ~
108
eswaribala@DESKTOP-55AGI0I:~$ pstree -p | grep "apache2"
init(1)-+-apache2(105)-+-apache2(107)-+-{apache2}(110)
        |              |              |-{apache2}(111)
        |              |              |-{apache2}(112)
        |              |              |-{apache2}(113)
        |              |              |-{apache2}(114)
        |              |              |-{apache2}(115)
        |              |              |-{apache2}(116)
        |              |              |-{apache2}(117)
        |              |              |-{apache2}(118)
        |              |              |-{apache2}(119)
        |              |              |-{apache2}(120)
        |              |              |-{apache2}(121)
        |              |              |-{apache2}(122)
        |              |              |-{apache2}(123)
        |              |              |-{apache2}(124)
        |              |              |-{apache2}(125)
        |              |              |-{apache2}(126)
        |              |              |-{apache2}(127)
        |              |              |-{apache2}(128)
        |              |              |-{apache2}(129)
        |              |              |-{apache2}(130)
        |              |              |-{apache2}(131)
        |              |              |-{apache2}(132)
        |              |              |-{apache2}(133)
        |              |              |-{apache2}(134)
        |              |              `-{apache2}(135)
        |              `-apache2(108)-+-{apache2}(138)
        |                             |-{apache2}(139)
        |                             |-{apache2}(140)
        |                             |-{apache2}(141)
        |                             |-{apache2}(142)
```

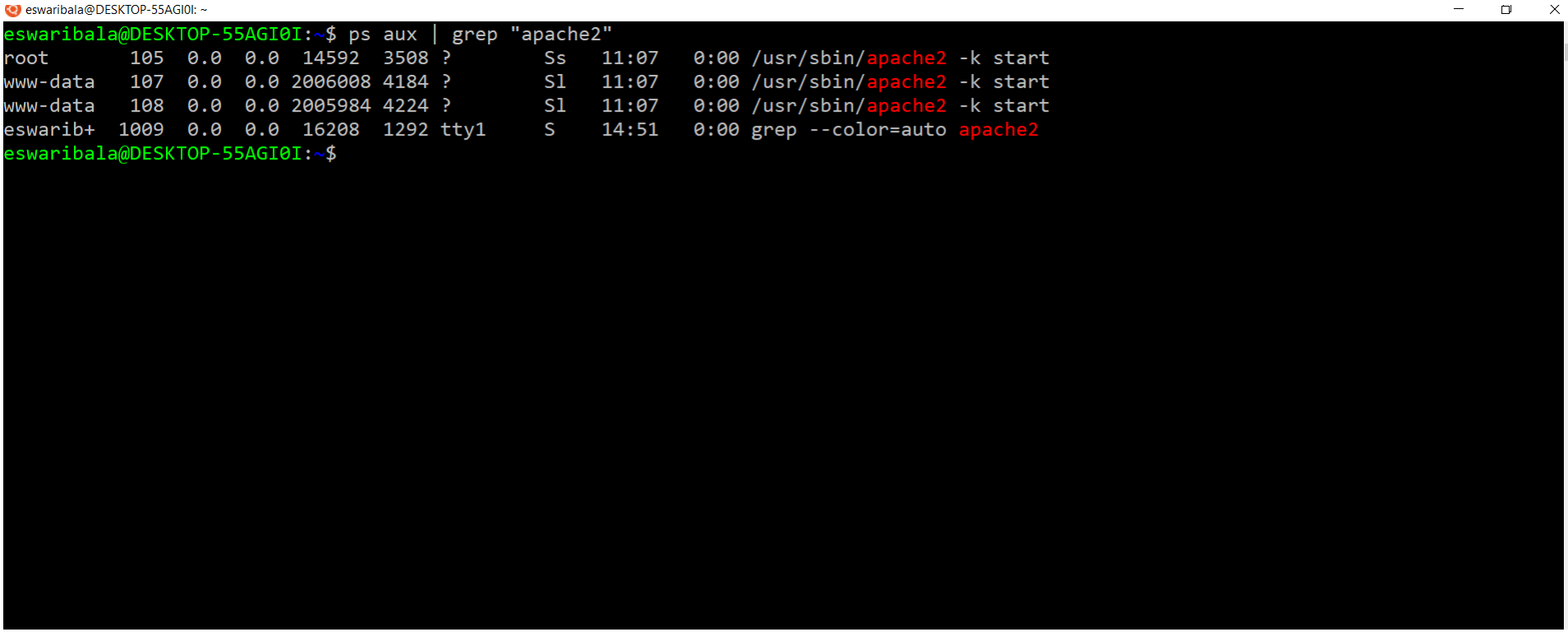
To get only the parent process
pstree -p | grep "apache2" | head -1

```
eswaribala@DESKTOP-55AGI0I: ~
eswaribala@DESKTOP-55AGI0I:~$
eswaribala@DESKTOP-55AGI0I:~$ pstree -p | grep "apache2" | head -1
init(1)-+-apache2(105)-+-apache2(107)-+-{apache2}(110)
eswaribala@DESKTOP-55AGI0I:~$
```

Method-4: How to find the Process ID (PID) of a program running on Linux using the ps Command

- The ps command displays information about a selection of the active processes.
- It displays the process ID (pid=PID), the terminal associated with the process (tname=TTY), the cumulated CPU time in [DD-]hh:mm:ss format (time=TIME), and the executable name (ucmd=CMD). Output is unsorted by default.

ps aux | grep "apache2"



Nano Useful Keys

- move forward one character: Ctrl+F (^F)
- move back one character: Ctrl+B (^B)
- move forward one word: Ctrl+Space (^Space)
- move back one word: Alt+Space (M-Space)
- move to the previous line: Ctrl+P (^P)
- move to the next line: Ctrl+N (^N)
- move to the next page: Ctrl+V (^V)
- move to the previous page: Ctrl+Y (^Y)
- move to the beginning of the line: Ctrl+A (^A)
- move to the end of the line: Ctrl+E (^E)

Nano Useful Keys

- **Search a Text File**
- To search for a particular word or part of a text inside the editor, use the “where is” option with the Ctrl+W shortcut (^W). This will open a search prompt where you can type in the text you want to find. To continue to the next result, use Alt+W (M-W).
- The search bar can also find specific line numbers. Press Ctrl+T (^T) while in it and the line number you want to find.

Nano Useful Keys

- Replace Text
- To replace text in the file, first open the search bar with Ctrl+W (^W) and then press Ctrl+R (^R). It will open a search bar to type in what you want to replace, as seen in the image below.

Select, Copy, Cut and Paste Text

- To select part of a file, navigate to the beginning of the text, press the Alt+A shortcut (M-A) and use the arrow keys to move over the text you wish to select.

- Next, you can copy the selected text with the Alt+6 combination (M-6) or cut with Ctrl+K (^K). If you use these shortcuts without selecting any text prior, it will copy or cut the entire line of text.

- To paste text, use Ctrl+U (displayed as ^U).

Save a File

- To save a file, use the Ctrl+O (^O) keyboard combination. It will ask you to enter a file name or confirm the name of an existing file.

Apache Performance Tuning: Swap Memory

- Before we get into the nitty-gritty of Apache tuning, we need to understand what happens when a VPS server or Dedicated server goes unresponsive due to a poorly optimized configuration.
- An over-tuned server is one that is configured to allow more simultaneous requests (ServerLimit) than the server's hardware can manage.
- Servers set in this manner have a tipping point, once reached, the server will become stuck in a perpetual swapping scenario.

Apache Performance Tuning: Swap Memory

- Meaning the Kernel is stuck rapidly reading and writing data to and from the system swap file.
- Swap files have read/write access speeds vastly slower than standard memory space.
- The swap files' latency causes a bottleneck on the server as the Kernel attempts to read and write data faster than is physically possible or more commonly known as thrashing.

Apache Performance Tuning: Swap Memory

- If not caught immediately, thrashing spirals the system out of control leading to a system crash.
- If trashing is left running for too long, it has the potential of physically harming the hard drive itself by simulating decades of read/write activity over a short period.
- When optimizing Apache, we must be cautious not to create a thrashing scenario.
- We can accomplish this by calculating the thrashing point of the server based on several factors.

Estimate the Thrashing Point

- Calculating the estimated thrashing point or ServerLimit of a server uses a simple equation:

(buff/cache – Reserved)/Avg.Apache

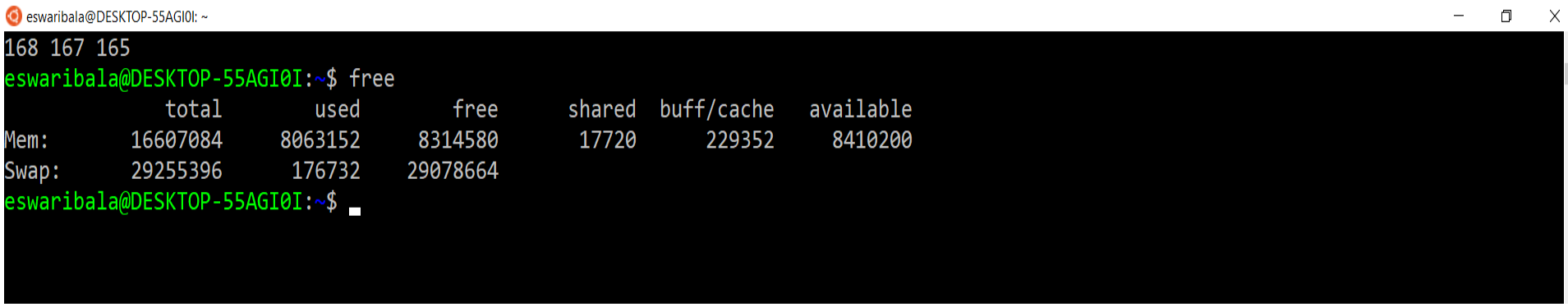
- buff/cache: The total memory used by the Kernel for buffers and cache.
- Reserved: The amount of memory reserved for non-Apache processes.
- Available: The difference between buff/cache and Reserved memory.
- Avg.Apache: The average of all running Apache children during peak operational hours.

Estimate the Thrashing Point

- The thrashing point value is equal to the number Apache children the server can run; this applies to either threaded or non-thread children.
- When the number of children running in memory meets the calculated thrashing point, the server will begin to topple.

Buffer/Cache Memory

- On modern Linux systems, the buffer/cache can be derived using the /proc/meminfo file by adding the Buffers, Cached and Slab statistics.
- Using the free command, we can quickly grab this information, as in the example below:



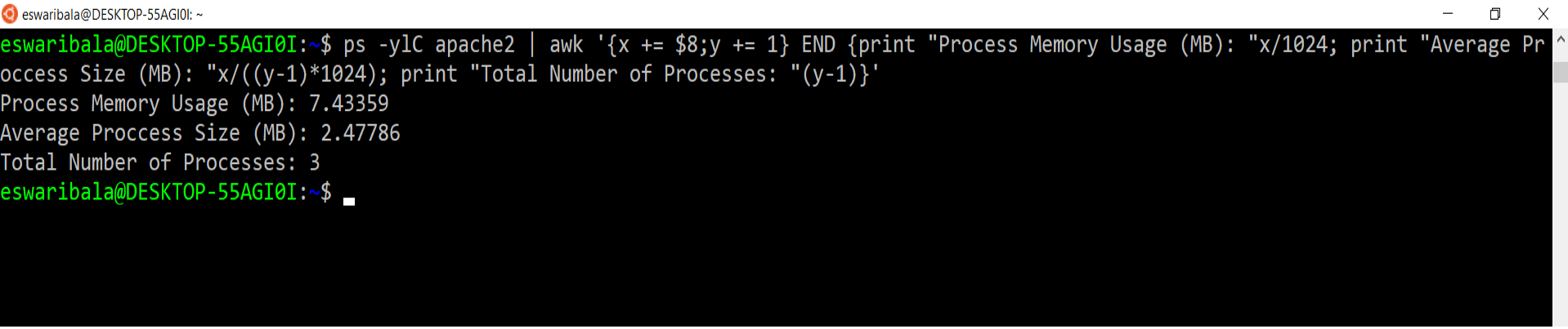
Reserved Memory

- Reserved memory is a portion of memory held for other services aside from Apache.
- Some of the biggest contenders for additional memory outside of Apache are MySQL, Tomcat, Memcache, Varnish, and Nginx.
- It is necessary to examine these services configs to determine a valid reserved memory.
-  Rule-of-Thumb:
- Save 25% of the total buff/cache memory for any extra services ran on the server.

Average Apache Memory

- Finding the average size of Apache processes is relatively simple using the ps command to list the RSS (Resident Set Size) of all running httpd processes.
- This example uses a short awk script to print out the average instead of listing the sizes.
- ps -ylC apache2 | awk '{x += $8;y += 1} END {print "Process Memory Usage (MB): "x/1024; print "Average Proccess Size (MB): "x/((y-1)*1024); print "Total Number of Processes: "(y-1)}'

Average Apache Memory



Calculate the Thrashing Point

- Once collected divided the Available memory by Avg. Apache, rounding down to the nearest whole number. Available memory is the buff/cache memory minus the Reserved memory.

Variable	KB	Notes
buffer/cache	708436	Gathered from the **free** command
Reserved	177109	**Rule-of-thumb:** up to 25% of **buffer/cache**
Available	531327	**buffer/cache** minus **Reserved**
Avg Apache	22200	Obtained using the **ps** command
Thrashing Point	23	**Available** divided by **Avg Apache** (round down)

The table summarizes how to derived each variable to calculate an estimated thrashing point.

Calculate the Thrashing Point

General Thrashing Point Estimates			
Memory	Requests	Memory	Requests
2 GB	23	12GB	140
4 GB	46	16 GB	187
6 GB	70	24 GB	281
8 GB	93	32 GB	374

Conservative estimates based on server memory, assuming ~80% of memory available and a base size for Apache children instance of 64k.

Apache Performance Tuning: MPM Modules

- The keystone for understanding Apache server performance is by far the Multiprocessing Modules (MPMs).
- These modules determine the basis for how Apache addresses multiprocessing.
- Multiprocessing means running multiple operations simultaneously in a system with multiple central processing units (CPU Cores).

Apache Performance Tuning: MPM Modules

- These modules are:
  - MPM Prefork
  - MPM Worker
  - MPM Event
  - Other MPMs

Apache Performance Tuning: MPM Modules

-  MPM Prefork

Avoid using MPM Prefork whenever possible. It’s inability to scale well with increased traffic will quickly outpace the available hardware on most system configurations.

Apache Performance Tuning: MPM Modules

-  MPM Worker

- A hybrid pre-forking, multi threaded, multiprocessing web server.
- In the same fashion as MPM Prefork, MPM Worker uses the same approach with a single master parent process governing all children within its server pool.
- MPM Worker has set the foundation for multi threaded multiprocessing in Apache servers which became stable in Apache 2.2.

Apache Performance Tuning: MPM Modules

-  MPM Worker

- The threaded configuration allows Apache to service hundreds of requests with ease while retaining only a dozen or so child processes in memory.
- The MPM Worker make for both a high capacity and low resource solution for web service.
- The KeepAliveTimeOut directive currently defines the amount of time Apache will wait for requests.
- When utilizing KeepAlive with MPM Worker use the smallest KeepAliveTimeout as possible (1 second preferably).

Apache Performance Tuning: MPM Modules

-  MPM Event

- Based off the MPM Worker source code, MPM Event shares configuration directives with MPM Worker.
- It works nearly identical to MPM Worker except when it comes to handling KeepAlive requests.
- MPM Event uses a dedicated Listener thread in each child process.
- This Listening thread is responsible for directing incoming requests to an available worker thread.
- The Listening thread solves the issue encountered by MPM Worker which locks entire threads into waiting for the KeepAliveTimeout.

Apache Performance Tuning: MPM Modules

-  MPM Event

- The Listener approach of MPM Event ensures worker threads are not “stuck” waiting for KeepAliveTimeout to expire.
- This method keeps the maximum amount of worker threads handling as many requests as possible.
- MPM Event is stable in Apache 2.4, older versions can use MPM Worker as an alternative

Apache Performance Tuning: MPM Modules

- Which MPM is the best?
- When considering optimization, it is essential to understand there is no such thing as a one-size-fits-all Apache configuration.
- Correctly choosing an MPM requires analysis of many moving variables like traffic, site code, server type, PHP Handler and available hardware.
- Every server is unique making the best MPM an entirely subjective choice.

Apache Performance Tuning: MPM Modules

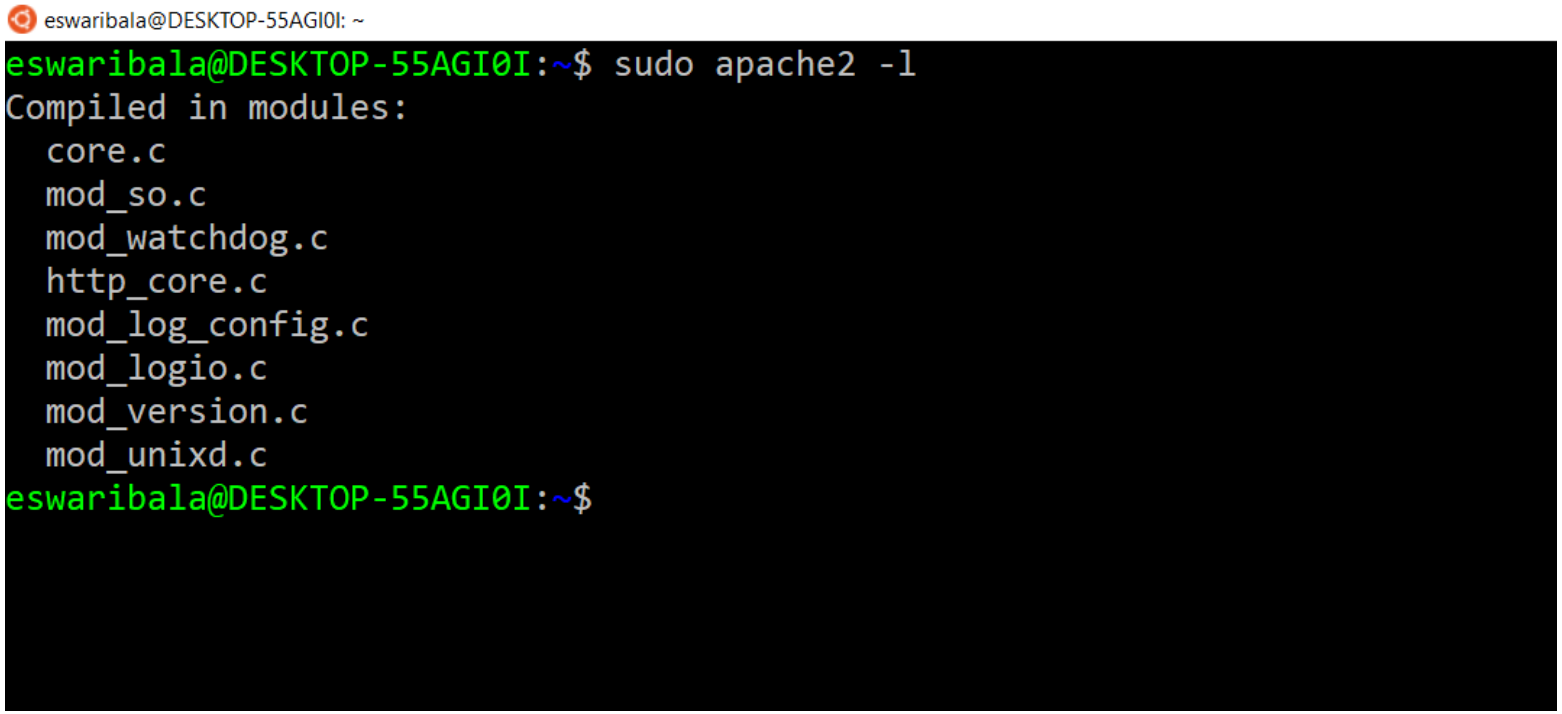
- If your application code does not support multi-threading, then your choice will inevitably be MPM Prefork purely on a compatibility basis.
- MPM Prefork includes software modules like mod_php (DSO).
- MPM Worker without KeepAlive performs very well if your application is a high-performance load balanced API system.
- The scalability and flexibility of MPM Event is a solid choice for hosting multiple small to medium sites in a shared hosting configuration.

Apache Performance Tuning: MPM Modules

- Most simple servers setups operate well under the self-governing default configuration of MPM Event, making it an ideal starting point for optimization tuning.
- Once chosen, an MPM can then move onto Configuration Directives to review which settings pertain to server performance and optimization.

View installed modules

- You can determine which MPM Apache2 is currently built with by executing apache2 -l



View installed modules

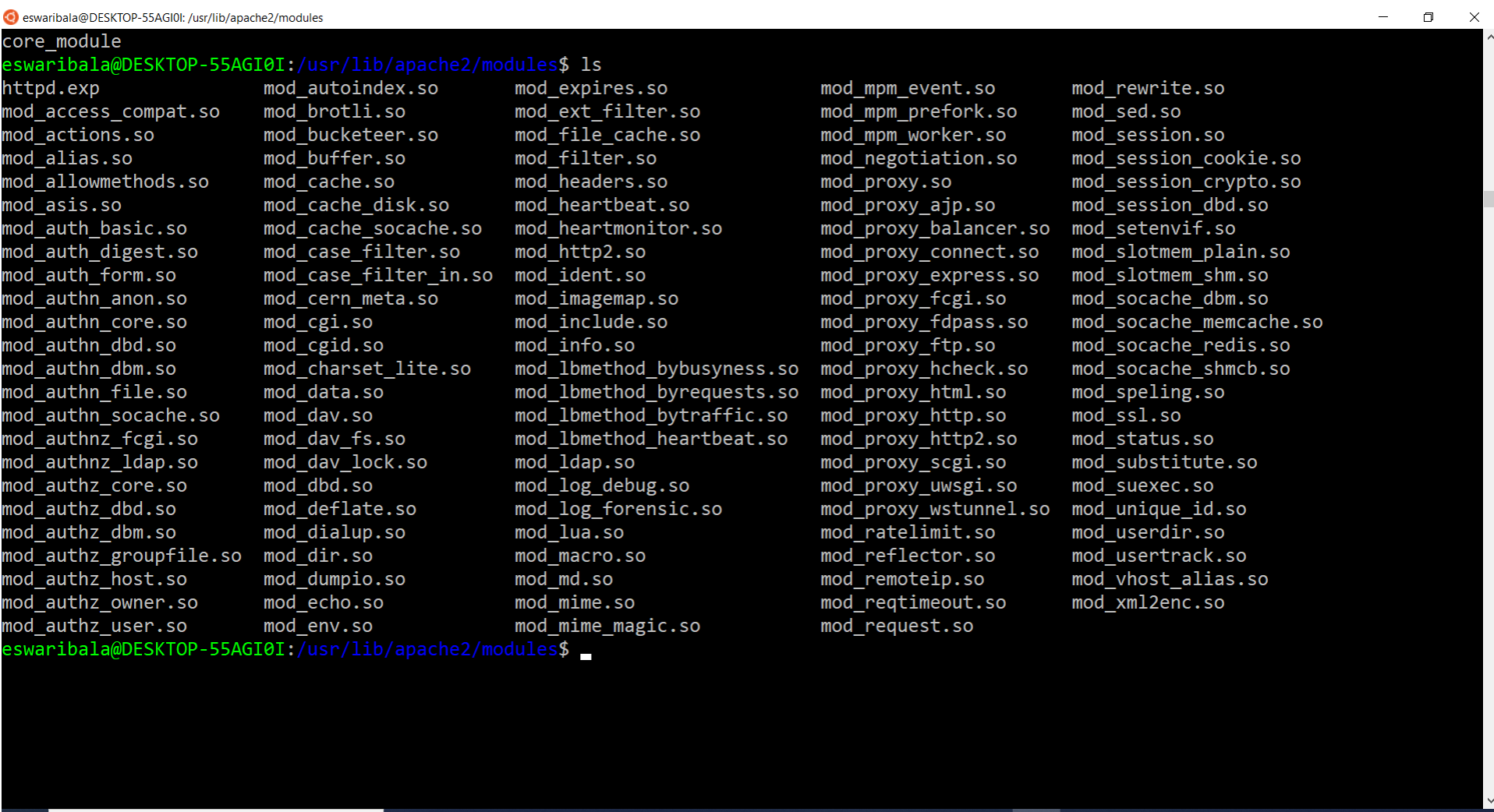
- This command will list all modules, both static and shared:
- MPMs can be built as static modules on all platforms.
- sudo apachectl -M

View installed modules

eswaribala@DESKTOP-55AGI0I: /

```
eswaribala@DESKTOP-55AGI0I:/$ sudo apache2ctl -M
[sudo] password for eswaribala:
Loaded Modules:
 core_module (static)
 so_module (static)
 watchdog_module (static)
 http_module (static)
 log_config_module (static)
 logio_module (static)
 version_module (static)
 unixd_module (static)
 access_compat_module (shared)
 alias_module (shared)
 auth_basic_module (shared)
 authn_core_module (shared)
 authn_file_module (shared)
 authz_core_module (shared)
 authz_host_module (shared)
 authz_user_module (shared)
 autoindex_module (shared)
 deflate_module (shared)
 dir_module (shared)
 env_module (shared)
 filter_module (shared)
 mime_module (shared)
 mpm_event_module (shared)
 negotiation_module (shared)
 reqtimeout_module (shared)
 setenvif_module (shared)
 status_module (shared)
eswaribala@DESKTOP-55AGI0I:/$
```

View installed modules



Apache Performance Tuning: MPM Modules

- **Enable/Disable Modules**

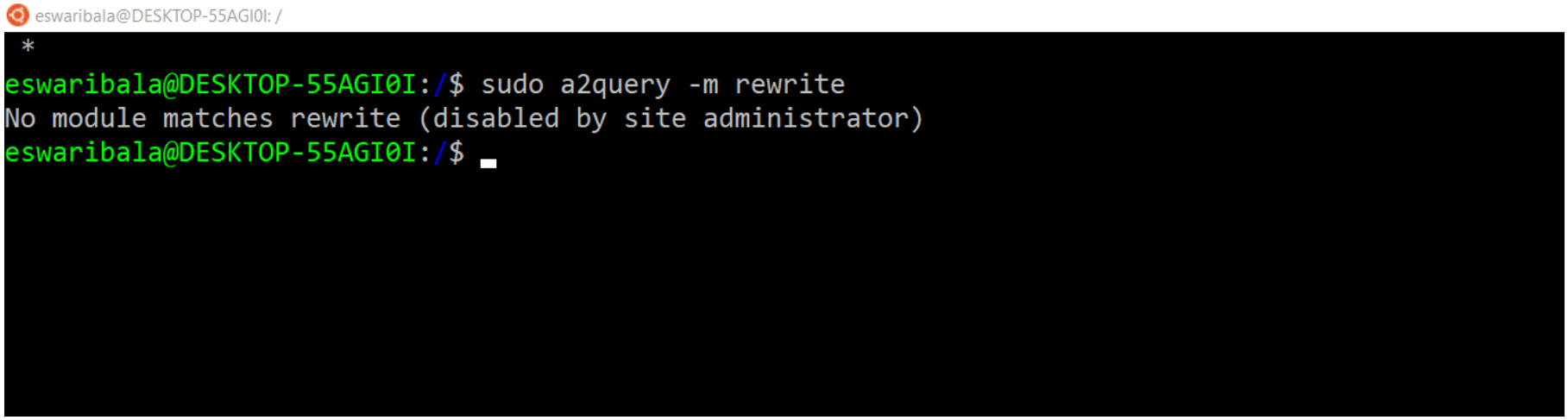
eswaribala@DESKTOP-55AGI0I: /

```
eswaribala@DESKTOP-55AGI0I:/$ sudo a2enmod rewrite
Module rewrite already enabled
eswaribala@DESKTOP-55AGI0I:/$ sudo service apache2 restart
 * Restarting Apache httpd web server apache2
sleep: cannot read realtime clock: Invalid argument

eswaribala@DESKTOP-55AGI0I:/$ sudo a2dismod rewrite
Module rewrite disabled.
To activate the new configuration, you need to run:
  service apache2 restart
eswaribala@DESKTOP-55AGI0I:/$ sudo service apache2 reload
 * Reloading Apache httpd web server apache2
 *
eswaribala@DESKTOP-55AGI0I:/$
```

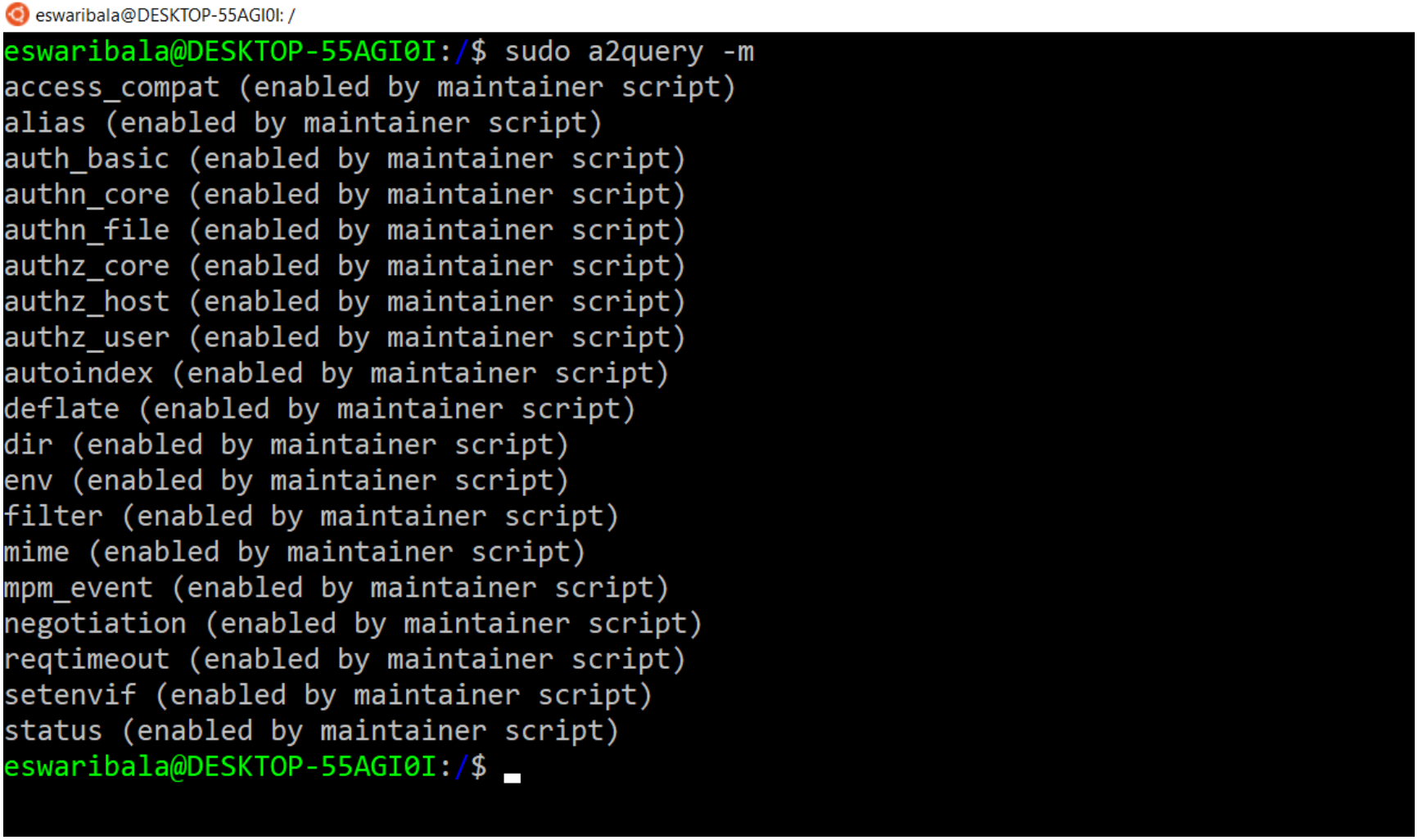
Apache Performance Tuning: MPM Modules

- **Check Modules Status**



Apache Performance Tuning: MPM Modules

- **List Apache2 Modules**



How To Configure Apache HTTP with MPM Event and PHP-FPM

- **Step 1 — Changing the Multi-Processing Module**
  - sudo service apache2 stop
  - `sudo a2dismod php7.2`
  - sudo a2dismod mpm_prefork
  - sudo a2enmod mpm_event
- **Step 2 — Configuring Apache HTTP to Use the FastCGI Process Manager**
  - sudo apt install php-fpm
  - sudo apt install libapache2-mod-fcgid (httpd to php commn lib)

How To Configure Apache HTTP with MPM Event and PHP-FPM

- **Step 2 — Configuring Apache HTTP to Use the FastCGI Process Manager**
  - sudo a2enconf php7.2-fpm
  - sudo a2enmod proxy
  - sudo a2enmod proxy_fcgi
  - `cat /etc/apache2/conf-enabled/php7.2-fpm.conf`
  - sudo apachectl configtest
  - sudo service apache2 restart

How To Configure Apache HTTP with MPM Event and PHP-FPM

- **Step 3 — Checking Your Configuration**
  - sudo apachectl -M | grep 'mpm’
  - sudo apachectl -M | grep 'proxy’
  - sudo nano /var/www/test.com/info.php
  - http://localhost:76/info.php

Apache Performance Tuning: MPM Directives

- **General Optimization**
- IfModule
  - An important directive to learn when working with Apache servers is the IfModule conditional statement.
  - There are two parts to the IfModule statement.
  - A beginning, which also accepts a module name or module source file name, as well as a closing statement.
  - When the provided module is loaded into Apache, then all directives between the beginning IfModule statement and the closing IfModule statement are also read into the Apache running configuration.

Apache Performance Tuning: MPM Directives

- <ifModule mpm_prefork_module>
- MaxSpareServers 16
- </ifModule>
- Timeout 60

Apache Performance Tuning: MPM Directives

- IfModule statements are used to maintain compatibility within Apache configuration between module changes.
- Maintaining compatibility is done by grouping directives into IfModule statements, so they are only used when the required module is loaded.
- Ensuring a syntactically correct configuration file even when swapping modules.
- Appropriately wrapping everything in an IfModule statement is a best practice standard with Apache and should be adhered to for superior compatibility in config files.

Apache Performance Tuning: MPM Directives

- Timeout
  - The numerical value of seconds Apache waits for all common I/O events.
  - Apache will abandon requests fail to complete before the provided Timeout value.
  - Determining the right Timeout depends on both traffic habits and hosted applications.
  - Ideally, Timeout should be as low as possible while still allowing the vast majority of regular traffic to operate without issue.
  - Large timeouts, those above 1 minute, open the server to SlowLoris style DOS attacks and foster a long wait in the browser when it encounters a problem.
  - Lower timeouts allow Apache to recover from errant stuck connections quickly.

Apache Performance Tuning: MPM Directives

- Timeout
  - It becomes necessary to strike a balance between the two extremes.

Tip:

Timeout	
Syntax	`Timeout number`
Default	*60*
Modules	*core/none*
Doc	*Timeout*

Avoid increasing the global *Timeout* when addressing issues with a single script, or user, that requires a long *Timeout*. Problems can usually be resolved by a .htaccess file or include file to increase the *Timeout* directive for that specific script.

Apache Performance Tuning: MPM Directives

- KeepAlive
- A simple on|off toggle enables the KeepAlive protocols with supported browsers.
- The KeepAlive feature can provide as much as a 50% reductions in latency, significantly boosting the performance of Apache.
- KeepAlive accomplishes this by reusing the same initial connections a browser creates when connecting to Apache for all follow-up requests which occur within a short period.

Apache Performance Tuning: MPM Directives

- KeepAlive is a powerful feature and in general, should be enabled in most situations.
- It works great for reducing some of the CPU and Network overhead with modern element heavy websites.
- For example, an easy way to visualize KeepAlive is with the "**hold the door**" phrase.
- Imagine a queue of people entering a building through a single doorway.
- Each person is required to open the door, walk through it, then close the door before the next person does the same process.
- Mostly, that's how Apache works without KeepAlive.
- When enabled, the door stays open until all the people in line are through the door before it closes again.

Apache Performance Tuning: MPM Directives

KeepAlive

KeepAlive	
Syntax	`KeepAlive on|off`
Default	*on*
Modules	*core/none*
Doc	*KeepAlive*

Apache Performance Tuning: MPM Directives

- MaxKeepAliveRequests
- Sets a limit on the number of requests an individual KeepAlive connection is permitted to handle.
- Once reached, Apache forces the connection to terminate, and creates a new one for any additional requests.
- Determining an ideal setting here is open to interpretation.
- Generally, you want this value to be at least as high as the largest count of elements (HTML, Text, CSS, Images, Etc..) served by the most heavily trafficked pages on the server.

Apache Performance Tuning: MPM Directives

Rule-of-Thumb:

MaxKeepAliveRequests	
Syntax	`MaxKeepAliveRequests number`
Default	*100*
Modules	*core/none*
Doc	*MaxKeepAliveRequests*

Set *MaxKeepAliveRequests* to double that of the largest count of elements on common pages. (Services like webpagetest.org or gtmetrix.com can count elements on a page).

Apache Performance Tuning: MPM Directives

- KeepAliveTimeout
- This directive is measured in seconds and will remain idle waiting for additional requests from its initiator.
- Since these types of connections are only accessible to their initiator, we want to keep KeepAliveTimeout very low.
- A low value prevents too many KeepAlive connections from locking out new visitors due to connection priority.

Apache Performance Tuning: MPM Directives

Tip:

KeepAliveTimeout	
Syntax	`KeepAliveTimeout number`
Default	*5*
Modules	*core/none*
Doc	*KeepAliveTimeout*

Event.

A large *MaxKeepAliveRequests* directive with a very low *KeepAliveTimeout* allows active visitors to reuse connections while also quickly recovering threads from idle visitors.

Configuration: Set *MaxKeepAliveRequests* to 500+, Set *KeepAliveTimeout* to 2

Requirements: Works best on MPM

MPM Event/Worker Optimization

- The two modules, MPM Event, and MPM Worker for most intents and purposes operate identically.
- The difference is apparent in the way each handles KeepAlive requests.
- The MPM Worker locks threads for the duration of the KeepAlive process and directly affects the number of available threads able to handle new requests.
- The MPM Event uses a Listener thread for each child.
- These Listener threads handle standard requests, and KeepAlive requests alike meaning thread locking will not reduce the capacity of the server.
- Without thread locking, MPM Event is the superior choice but only in Apache 2.4.
- Before Apache 2.4 the MPM Event was unstable and prone to problems.

MPM Event/Worker Optimization

- **ServerLimit**
- ServerLimit represents the upper limit of children Apache is allowed.
- The practical usage for ServerLimit is creating a hard ceiling in Apache to protect against input errors with MaxRequestWorkers.
- The cap prevents spawning vastly more children than a system can handle, resulting in downtime, revenue loss, reputation loss or even data loss.

MPM Event/Worker Optimization

- **ServerLimit**
- ServerLimit ties in directly with the thrashing point.
- The thrashing point is the maximum number of children Apache can run before memory usage tips the scale into perpetual swap.
- Match the ServerLimit to the calculated thrashing point to safeguard the server.
-

ServerLimit	
Syntax	`ServerLimit number`
Default	*16*
Modules	*mpm_event module* *mpm_worker_module*
Doc	*ServerLimit*

MPM Event/Worker Optimization

- **ThreadsPerChild**
- Used to define the limit of threads that each Apache child can manage.
- Every thread running can handle a single request.
- The default of 25 works well for most cases and is a fair balance between children and threads.
- There is an upper limit on this directive as well, and the limit is controlled by the ThreadLimit directive, which defaults to 64 threads.
- The adjustments to increase ThreadsPerChild past 64 threads also need to be made to ThreadLimit.

MPM Event/Worker Optimization

- **ThreadsPerChild**
- Increasing this value allows each child to handle more requests keeping memory consumption down while allowing a larger MaxRequestWorkers directive.
- A key benefit of running more threads within each child is shared memory cache access.
- Threads from one child cannot access caches from another child.
- Boosting the number of threads per child squeezes out more performance due to this sharing of cache data.
- The major downside for increased threads per child occurs during child recycling.
- The capacity of the server is diminished by the number of threads configured for each child when that child process is eventually recycled (graceful restart).

MPM Event/Worker Optimization

- **ThreadsPerChild**

ThreadsPerChild	
Syntax	`ThreadsPerChild number`
Default	*25*
Modules	*mpm_event module* *mpm_worker_module*
Doc	*ThreadsPerChild*

CPU Cores	MPM Event/Worker				
	ServerLimit	x	ThreadsPerChild	=	MaxRequestWorkers
2	2	x	64	=	128
4	4	x	50	=	200
8	8	x	25	=	200
16	16	x	25	=	400
32	32	x	16	=	512

Examples of various *ThreadsPerChild* configurations in Worker based MPMs.

MPM Event/Worker Optimization

- **ThreadsPerChild**
- Inversely the opposite reaction is achieved by lowering ThreadsPerChild.
- Fewer threads per child require more children to run an equal amount of MaxRequestWorkers.
- Since children are full copies of Apache, this increases Apache's overall memory footprint but reduces the impact when recycling children.
- Fewer threads mean fewer potential "stuck" threads during the recycle procedure, keeping the higher capacity of requests available overall children.
- Having fewer threads per child provides increased shared memory isolation.
- For instance, dropping ThreadsPerChild to 1 gives the same request isolation of MPM Prefork but also inherits its massive performance tax as well, requiring one child per one request.

MPM Event/Worker Optimization

Tip:

When setting *ThreadsPerChild* always consider the server environment and hardware.

- A memory-heavy shared server hosting numerous independent accounts might opt for a lower *ThreadsPerChild*, reducing the potential impact of one user affecting another.
- A dedicated Apache server in a high capacity load balanced configuration can choose to increase *ThreadsPerChild* significantly for a better overall performance of each thread.

MPM Event/Worker Optimization

- **ThreadLimit**
- Used to set the maximum value of ThreadsPerChild.
- This directive is a hard ceiling for ThreadsPerChild.
- It helps protect against typographical errors with the ThreadLimit.
- ThreadsPerChild directive which could quickly spin a server out of control if too many threads are allowed due to an input error.
- This setting need to be adjusted in some high-end servers when the system needs more than the default of 64 threads per child.

MPM Event/Worker Optimization

- **ThreadLimit**

ThreadLimit	
Syntax	`ThreadLimit number`
Default	*64*
Modules	*mpm_event module* *mpm_worker_module*
Doc	*ThreadLimit*

MPM Event/Worker Optimization

- **MaxRequestWorkers / MaxClients**
- The directive sets the limit for active worker threads across all running children and acts as a soft ceiling with ServerLimit taking control as the hard limit.
- When the number of total running threads has reached or exceeded MaxRequestWorkers, Apache no longer spawns new children.MaxRequestWorkers/MaxClients.
- Determining the MaxRequestWorkers is a critical part of server optimization.
- An optimal setting is based on several changing variables.

MPM Event/Worker Optimization

- **MaxRequestWorkers / MaxClients**
- This means its configuration needs to be reevaluated and tailored periodically over time, changed by watching traffic habits and system resource usage.
- The Apache status Scoreboard is an effective tool for analysis of Apache performance.
- It is typical of Worker based MPM systems to run an isolated third-party PHP handler like Mod_fcgid, PHP-FPM, and mod_lsapi.

MPM Event/Worker Optimization

- **MaxRequestWorkers / MaxClients**
- These modules are responsibleMPM Event/Worker2 for processing PHP code outside of Apache and frees up Apache to handle all other non-PHP requests such as HTML, TEXT, CSS, Images, etc…
- These requests are far less taxing on server resources which allows Apache to handle larger volumes of requests, such as those beyond 400 MaxRequestWorkers.

MPM Event/Worker Optimization

- **MaxRequestWorkers / MaxClients**

CPU Cores	MPM Event/Worker				
	ServerLimit	x	ThreadsPerChild	=	MaxRequestWorkers
2	2	x	64	=	128
4	4	x	50	=	200
8	8	x	50	=	400
16	16	x	32	=	512
32	32	x	32	=	1024

Examples of various *MaxRequestWorkers* configurations using Worker based MPM

MPM Event/Worker Optimization

- **MinSpareThreads**
- The least number of Threads that should remain open, waiting for new requests.
- MinSpareThreads is a multiple of ThreadsPerChild and cannot exceed MaxSpareThreads, though it can match it.

Rule-of-Thumb:

Set *MinSpareThreads* to equal 50% of *MaxRequestWorkers*.

MPM Event/Worker Optimization

- **MinSpareThreads**
- Spare threads are idle workers threads.
- These threads are merely waiting for new incoming requests and are governed by the Apache child process that spawned them.
- If there are less available threads than MinSpareThreads, the Apache parent will generate a new child with another ThreadsPerChild worth of threads.

MPM Event/Worker Optimization

- **MinSpareThreads**

CPU Cores	MPM Event/Worker				
	ServerLimit	x	ThreadsPerChild	=	MaxRequestWorkers
2	2	x	64	=	128
4	4	x	50	=	200
8	8	x	50	=	400
16	16	x	32	=	512
32	32	x	32	=	1024

Examples of various *MaxRequestWorkers* configurations using Worker based MPM

MPM Event/Worker Optimization

- **MaxSpareThreads**
- This directive governs the total number of idle threads allowed on the server across all children.
- Any threads above this limit direct their parent to shut down to reduce memory consumption during off-peak hours.

MaxSpareThreads	
Syntax	`MinSpareThreads` *number***
Default	*250*
Modules	*mpm_event_module* *mpm_worker_module*
Doc	*MinSpareThreads*

*** This value must be evenly divisible by **ThreadsPerChild**. If not, Apache will force it to the closest multiple, rounding down.*

MPM Event/Worker Optimization

- **MaxSpareThreads**
- This directive governs the total number of idle threads allowed on the server across all children.
- Any threads above this limit direct their parent to shut down to reduce memory consumption during off-peak hours.

MaxSpareThreads	
Syntax	`MinSpareThreads` *number***
Default	*250*
Modules	*mpm_event_module* *mpm_worker_module*
Doc	*MinSpareThreads*

*** This value must be evenly divisible by **ThreadsPerChild**. If not, Apache will force it to the closest multiple, rounding down.*

MPM Event/Worker Optimization

- **MaxSpareThreads**
- Having a limit to the number of idle open threads is excellent for smaller servers with hardware constraints.
- However, it mostly unneeded on today's modernizing hardware.

MPM Event/Worker Optimization

- **MaxSpareThreads**

Tip:

Configuring Apache as an open throttle is a high-performance configuration for servers with significant RAM and multiple CPU cores. When running the open throttle configuration, all available threads become available at all time. Apache's memory usage will stay near its peak at all times, a side effect due to running all the configured children into memory preemptively. This configuration will produce the best possible response times from Apache by maintaining persistent open connections ready to do work and removing the overhead of spawning new processes in response to traffic surges.

Configuration: Match both *MinSpareThreads* and *MaxSpareThreads* to *MaxRequestWorkers*.

Requirements: Make sure there is enough server RAM to run all *MaxRequestWorkers* at once.

MPM Event/Worker Optimization

- **StartServers**
- This directive governs the initial amount of children the Apache Parent process spawns when the Apache service is started or restarted.
- This is commonly left unchanged since Apache continuously checks the current running children in conjunction with ThreadsPerChild and compare it to MinSpareThreads to determine if more children get forked.
- This process is repeated perpetually, with a doubling of new children on each iteration, until MinSpareThreads is satisfied.

MPM Event/Worker Optimization

- **StartServers**

Rule-of-Thumb:

Manually calculating StartServers is done by dividing *MaxRequestWorkers* by *ThreadsPerChild*, rounding down to the nearest whole number. This process forces all children to be created without delay at startup and begins handling requests immediately. This aspect is especially useful in modern Apache servers which require periodic restarts to load in directive changes.

StartServers	
Syntax	`StartServer number`
Default	*3*
Modules	*mpm_event module* *mpm_worker_module*
Doc	*StartServers*

MPM Event/Worker Optimization

- **MaxConnectionsPerChild / MaxRequestsPerChild**
- The number of requests a single Apache child process can handle equals a cumulative total on the child server across all threads it controls.
- Each request handled by a thread counts toward this limit to its parent.
- Once the child server has reached its limit, the child is then recycled.
- This directive is a stop-gap for accidental memory leaks.
- Some code executed through Apache threads may contain memory leaks.
- Leaked memory are portions of memory that subprocess failed to release properly, so they are inaccessible to any outside processes.

MPM Event/Worker Optimization

- **MaxConnectionsPerChild / MaxRequestsPerChild**
- The longer a leaking program is left running, the more memory it will leak.
- Setting a MaxConnectionsPerChild limit is a specific method for assuring Apache is periodically recycling programs to reduce the impact of leaked memory on the system.
- When using external code handlers like Mod_fcgid, PHP-FPM or mod_lsapi, it becomes necessary to set MaxConnectionsPerChild to 0 (unlimited), doing so prevents periodic error pages caused by Apache terminating threads prematurely.

MPM Event/Worker Optimization

- **MaxConnectionsPerChild / MaxRequestsPerChild**

Rule-of-Thumb:

MaxConnectionsPerChild/MaxRequestsPerChild	
Syntax	`MaxConnectionsPerChild number`
Default	*0 (unlimited)*
Modules	*mpm_event_module* *mpm_worker_module*
Doc	*MaxRequestWorkers*
***MaxConnectionsPerChild** replaced **MaxRequestsPerChild** with Apache 2.4*	

If the server encounters a memory leak never set the *MaxConnectionsPerChild / MaxRequestsPerChild* too low, instead start with 10,000 and reduce it incrementally.

MPM Prefork Optimization

- This MPM Prefork section details the use and performance considerations for various directives when running this module.
- This MPM is a non-threaded multi-processor designed for compatibility.
- It consists of a single Apache parent process, which is used to govern all new Apache processes also known as children.
- The following directives show how Apache is capable of performance tuning when using MPM Prefork.
- Unlike Worker based MPMs, optimizing MPM Prefork is generally simple and straightforward.

MPM Prefork Optimization

- There is a 1:1 ratio of Apache processes to incoming requests.
- However, MPM Prefork does not scale well with hardware and the more traffic it encounters, the more hardware it will need to keep up with the pace.
- It should be noted that some directives behave differently based on which MPM is loaded.

MPM Prefork Optimization

- **MaxRequestWorkers / MaxClients**
- Used to control the upper limit of children that the Apache parent server is allowed to have in memory at one time.
- These children (also called workers) handle requests on a 1:1 ratio.
- This translates into the maximum number of simultaneous requests the server can handle.
- MaxRequestWorkers / MaxClientsIf this directive is too low, Apache under-utilizes the available hardware which translates to wasted money and long delays in page load times during peak hours.

MPM Prefork Optimization

- **MaxRequestWorkers / MaxClients**
- Alternatively, if this directive is too high, Apache outpaces the underlying hardware sending the system into thrashing (link to thrashing article) scenario which can lead to server crashes and potential data loss.

MaxRequestWorkers/MaxClients	
Syntax	`MaxRequestWorkers/MaxClients` *number*
Default	*256*
Modules	*mpm_event_module* *mpm_worker_module*
Doc	*MaxRequestWorkers*
MaxConnectionsPerChild *replaced* ***MaxRequestsPerChild*** *with Apache 2.4*	

MPM Prefork Optimization

- **MinSpareServers**
- This directive defines a minimum number of spare children the Apache parent process can maintain in its memory.
- An additional server is a preforked idle Apache child that is ready to respond to a new incoming request.
- Having idle children waiting for new requests is essential for providing the fastest server response times.
- When the total idle children on the server drop below this value, a new child is preforked at the rate of one per second until this directive is satisfied.

MPM Prefork Optimization

- **MinSpareServers**
- The “one per second” rule is in place to prevent surges of the creation process that overload the server, however, this failsafe comes at a cost.
- The one per second spawn rate is particularly slow when it comes to handling page requests.
- So it’s highly beneficial to make sure enough children are preforked and ready to handle incoming requests.
-

MPM Prefork Optimization

- **MinSpareServers**

Rule of Thumb:

Never set this to zero. Setting this to 25% of *MaxRequestWorkers* ensures plenty resources are ready and waiting for requests.

MinSpareServers	
Syntax	`MaxConnectionsPerChild number`
Default	*0 (unlimited)*
Modules	*mpm_event module* *mpm_worker_module*
Doc	*MaxRequestWorkers*

MPM Prefork Optimization

- **MaxSpareServers**
- MaxSpareServers controls the maximum number of idle Apache child servers running at one time.
- An idle child is one which is not currently handling a request but waiting for a new request.
- When there are more than MaxSpareServers idle children, Apache kills off the excess.
- If the MaxSpareServers value is less than MinSpareServers, Apache will automatically adjust MaxSpareServers to equal MinSpareServers plus one.
- Like with MinSpareServers, this value should always be altered with available server resources in mind.

MPM Prefork Optimization

- **MaxSpareServers**

Rule of Thumb:

MaxSpareServers	
Syntax	`MaxSpareServers number`
Default	*10*
Modules	*mpm_prefork_module*
Doc	*MaxSpareServers*

Set this to double the value of *MinSpareServers*.

MPM Prefork Optimization

- **MaxSpareServers**
- Configuring Apache as an open throttle is a high-performance configuration for servers with significant RAM and multiple CPU cores.
- When running the open throttle configuration, all available Apache children become available at all times.
- As a side effect of running open throttle, the Apache memory usage will stay near its peak at all times, due to running all the configured children into memory preemptively.
- This configuration will produce the best possible response times by maintaining persistent open connections.

MPM Prefork Optimization

- **MaxSpareServers**
- Furthermore, in response to traffic surges, it removes the overhead that comes from spawning new processes.
- Configuration: Match both MinSpareServers and MaxSpareServers to MaxRequestWorkers.
- Requirements: Make sure there is enough server RAM to run all MaxRequestWorkers at once.

MPM Prefork Optimization

- **StartServers**
- Created at startup, are the initial amount of Apache child servers.
- This seldom changed directive only impacts Apache startup and restart processes.
- Generally not altered because Apache uses internal logic to work out how many child servers should be running.
- Many modern servers periodically restart Apache to address configuration changes, rotate log files or other internal processes.
- When this occurs during a high load traffic surge, every bit of downtime matters.
- You can manually set the StartServers directive to mirror that of your MinSpareServers to shave off time from the Apache startup.

MPM Prefork Optimization

- **StartServers**

Rule of Thumb:

The *StartServers* directive should mirror that of MinSpareServers.

StartServers	
Syntax	`MaxSpareServers number`
Default	*5*
Modules	*mpm_event_module* *mpm_worker_module*
Doc	*StartServers*

MPM Prefork Optimization

- **ServerLimit**
- The ServerLimit directive represents the upper limit of MaxRequestWorkers.
- This setting is generally used as a safeguard or ceiling against input errors when modifying MaxRequestWorkers.
- ServerLimit Default Setting.
- It becomes necessary to adjusted ServerLimit when the server is expected to handle more than the default of 256 requests simultaneously.
- ServerLimit ties in directly with the thrashing point.
- The thrashing point is the maximum number of children Apache can run before memory usage tips the scale into perpetual swap.
- Match the ServerLimit to the calculated thrashing point to safeguard the server.

MPM Prefork Optimization

- **ServerLimit**

ServerLimit	
Syntax	`ServerLimit number`
Default	*256*
Modules	*mpm_prefork_module*
Doc	*ServerLimit*

MPM Prefork Optimization

- **ServerLimit**

Note:

Increasing *ServerLimit* is not recommended with MPM Prefork. Running more than 256 simultaneous requests is hardware intensive when using the MPM Prefork module.

MPM Prefork Optimization

- **MaxConnectionsPerChild / MaxRequestsPerChild**
- This directive equals the number of requests a single Apache child server can handle.
- This directive is a stop-gap for accidental memory leaks.
- Code executed through Apache may contain faults which leak memory.
- These leaks add up over time making less and less of the shared memory pool of the child usable.
- The way to recover from leaked memory is to recycle the affected Apache child process.
- Setting a MaxConnectionsPerChild limit will protect from this type of memory leakage.

MPM Prefork Optimization

- **MaxConnectionsPerChild / MaxRequestsPerChild**

Note:

MaxConnectionsPerChild/MaxRequestsPerChild	
Syntax	`MaxConnectionsPerChild number`
Default	*0 (unlimited)*
Modules	*mpm_event_module* *mpm_worker_module*
Doc	*MaxRequestWorkers*
***MaxConnectionsPerChild** replaced **MaxRequestsPerChild** with Apache 2.4*	

Rule-of-Thumb: Never set this too low. If the server encounters memory leak issues start with 10,000 and reduce incrementally.

Apache Performance Tuning: Configuring MPM Directives

- On Ubuntu servers, Apache configuration files are located in /etc/apache2/
- Backup existing apache2.conf file:
  - cp -p /etc/apache2/apache2.conf{,.bak.$(date +%F_%H%M%S)}
- ls -lah /etc/apache2/apache2.conf*
- Open file for editing with your favorite editor:
- vim /etc/apache2/apache2.conf

Apache Performance Tuning: Configuring MPM Directives

- Append the necessary directive changes to the very bottom of the config file.

MPM Prefork Example:

```
Timeout 30
<IfModule mpm_prefork_module>
KeepAlive On
MaxKeepAliveRequests 500
KeepAliveTimeout 3


ServerLimit 23
StartServers 12

MinSpareServers 12
MaxSpareServers 23

MaxRequestWorkers 23
MaxConnectionsPerChild 10000
</IfModule>
```

MPM Event Example:

```
Timeout 30
<IfModule mpm_event_module>
KeepAlive On
KeepAliveTimeout 3
MaxKeepAliveRequests 500

ThreadsPerChild 25
ServerLimit 23
MaxRequestWorkers 400

StartServers 16
MinSpareThreads 200
MaxSpareThreads 400

MaxRequestsPerChild 10000
</IfModule>
```

MPM Worker Example:

```
Timeout 30
<IfModule mpm_worker_module>
KeepAlive On
KeepAliveTimeout 1
MaxKeepAliveRequests 500

ThreadsPerChild 25
ServerLimit 23
MaxRequestWorkers 400

StartServers 16
MinSpareThreads 200
MaxSpareThreads 400

MaxRequestsPerChild 10000
</IfModule>
```

Apache Performance Tuning: Configuring MPM Directives

- sudo apachectl –t
- Sudo service apache2 start
- Sudo apachectl start

How To Configure Apache to Use Custom Error Pages on Ubuntu

- **Creating Your Custom Error Pages**
- Go to /var/www/html
- echo "<h1 style='color:red'>Error 404: Not found :-(</h1>" | sudo tee /var/www/html/custom_404.html
- echo " I have no idea where that file is, sorry. Are you sure you typed in the correct URL? " | sudo tee -a /var/www/html/custom_404.html
- echo "<h1>Oops! Something went wrong...</h1>" | sudo tee /var/www/html/custom_50x.html
- echo " We seem to be having some technical difficulties. Hang tight. " | sudo tee -a /var/www/html/custom_50x.html

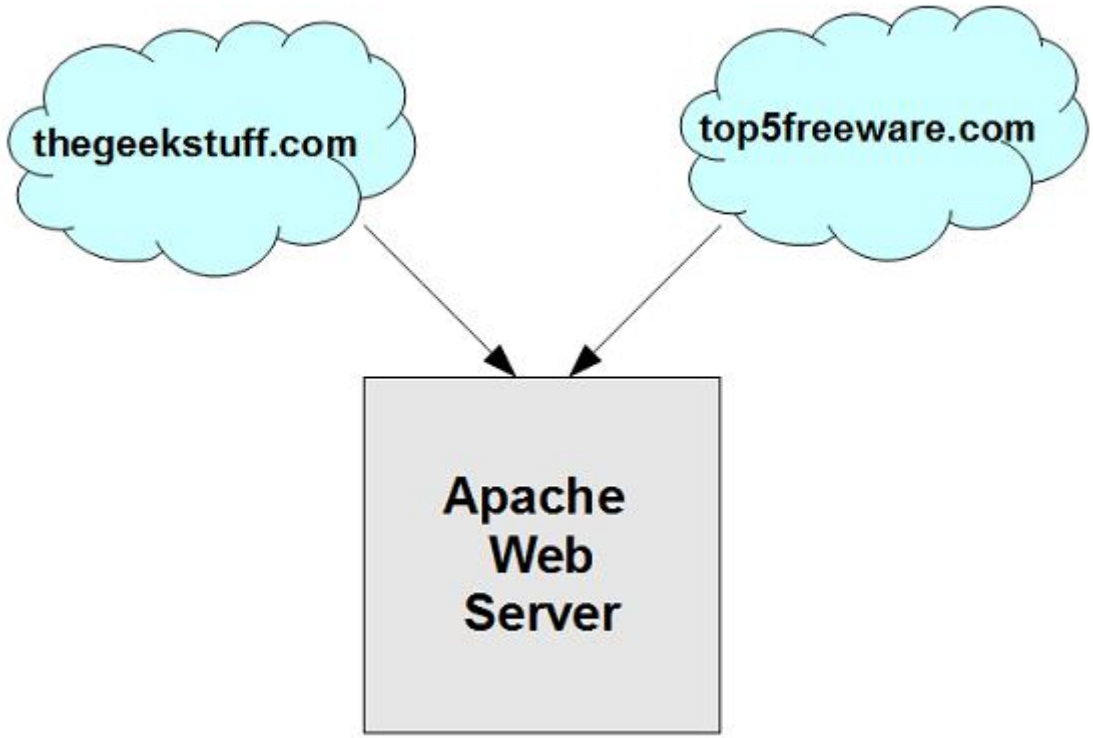
How To Configure Apache to Use Custom Error Pages on Ubuntu

- **Configuring Apache to Use your Error Pages**
- sudo nano /etc/apache2/sites-enabled/test.com.conf
- Refer for the changes
- sudo a2ensite test.com
- Sudo service apache2 reload

How To Setup Apache Virtual Host Configuration

- Using Apache Virtual Host, you can run several websites on the same server.

- For example, I can run both thegeekstuff.com and top5freeware.com on a single physical server that has one Apache webserver running on it.



How To Setup Apache Virtual Host Configuration

- There are two types of Apache virtual host configurations:
- 1) IP-Based Virtual Host and 2) Name-based Virtual Host.
- Name-based virtual host is recommended for most scenarios.

How To Setup Apache Virtual Host Configuration

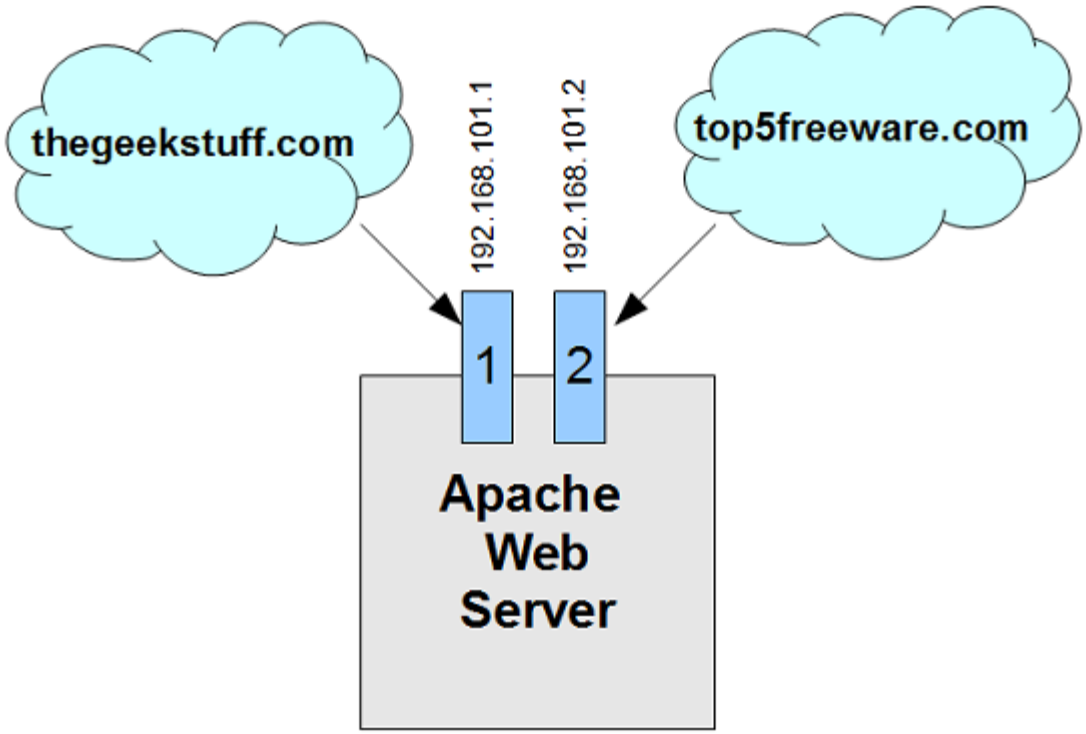
- **IP-Based Virtual Host**
- In this configuration, when you are pointing two websites (with different ip-address) to the server that runs Apache, that physical server should have two different ip-address configured.
- This means that the server should have two ethernet cards, each one of them configured to the ip-address of the corresponding website that Apache virtual host will be serving.
- So, this is not practical for most aspects, and you should not be using this.

How To Setup Apache Virtual Host Configuration

- **IP-Based Virtual Host**
- In the following example, the server contains two NIC cards, one is configured with 192.168.101.1 ip-address for thegeekstuff.com, another is configured with 192.168.102.1 for top5freeware.com.
- Both these ip-address are served by a single Apache webserver running on that server using IP-Based virtual host.

How To Setup Apache Virtual Host Configuration

- **IP-Based Virtual Host**



How To Setup Apache Virtual Host Configuration

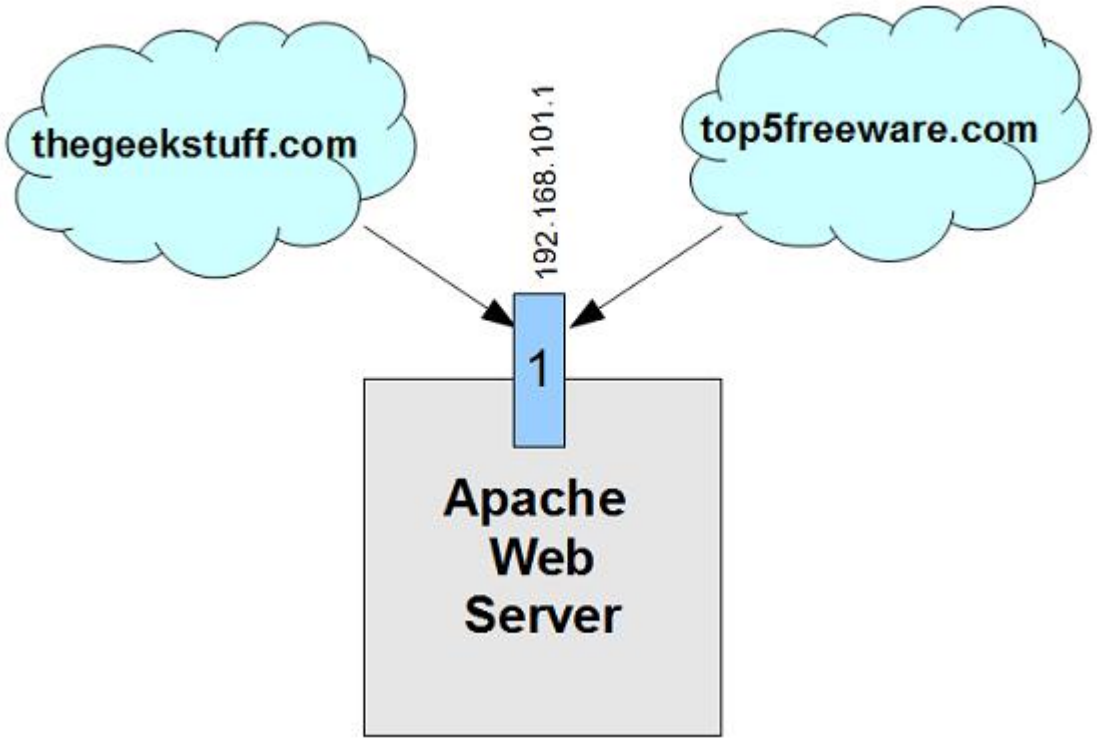
- **Name-Based Virtual Host**
- In this configuration, when Apache webserver receives a request, it looks for the hostname in the HTTP header, and depending on the hostname, it servers different websites.
- This is very easy, as you need only one ip-address on that physical server; but, you update the DNS with multiple website names pointing to the same ip-address.
- For all practical purpose, you'll be using only Name-based virtual host configuration.

How To Setup Apache Virtual Host Configuration

- **Name-Based Virtual Host**
- In the following example, the server contains only one NIC card, which is configured with 192.168.101.1 ip-address.
- The DNS entry for both thegeekstuff.com and top5freeware.com website points to 192.168.101.1 ip-address.
- When Apache recives a request, it looks for the hostname entry in the HTTP header, and serves the corresponding website.

How To Setup Apache Virtual Host Configuration

- **Name-Based Virtual Host**



How To Setup Apache Virtual Host Configuration

- sudo mkdir -p /var/www/geekstuff.com/public_html
- sudo mkdir -p /var/www/top5freeware.com/public_html
- sudo chown -R $USER:$USER /var/www/geekstuff.com/public_html
- sudo chown -R $USER:$USER /var/www/top5freeware.com/public_html
- sudo chmod -R 755 /var/www

How To Setup Apache Virtual Host Configuration

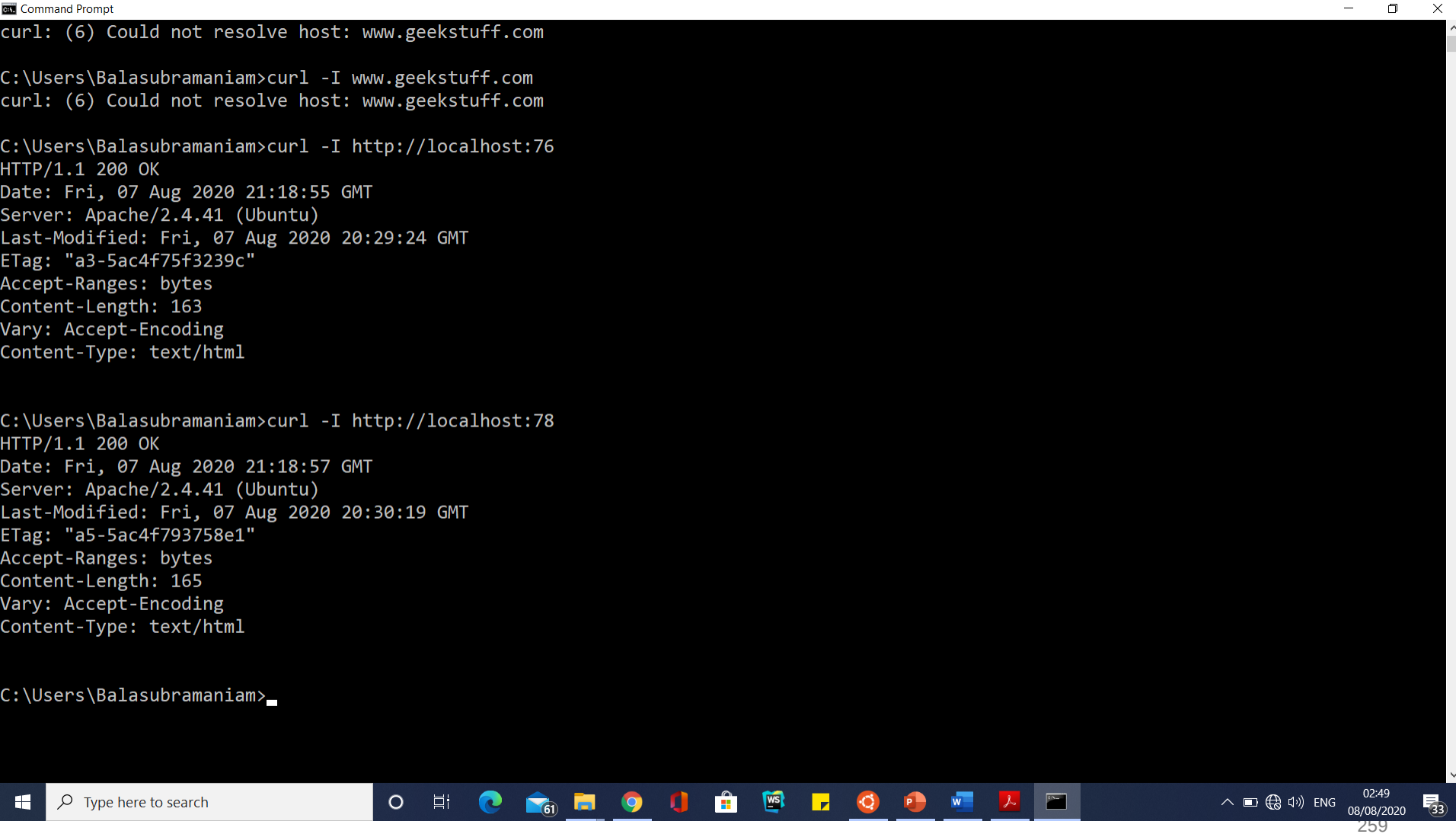
- sudo nano /var/www/geekstuff.com/public_html/index.html
- sudo nano /var/www/top5freeware.com/public_html/index.html

- sudo cp /etc/apache2/sites-available/000-default.conf /etc/apache2/sites-available/geekstuff.com.conf
- sudo nano /etc/apache2/sites-available/top5free.com.conf

How To Setup Apache Virtual Host Configuration

- sudo a2ensite geekstuff.com
- `sudo a2ensite top5free.com`
- `Sudo service apache2 reload`
- `sudo nano /etc/hosts`

How To Setup Apache Virtual Host Configuration



Apache 2 Security

- **Step 1 — Installing the Apache Utilities Package**
- sudo apt update
- sudo apt install apache2-utils
- **Step 2 — Creating the Password File**
- sudo htpasswd -c /etc/apache2/.htpasswd eswari
- sudo htpasswd /etc/apache2/.htpasswd another_user
- cat /etc/apache2/.htpasswd

Apache 2 Security

- **Step 3 — Configuring Apache Password Authentication**
- **Option 1: Configuring Access Control within the Virtual Host Definition (Preferred)**
- sudo nano /etc/apache2/sites-enabled/default-ssl.conf
- Refer deafultssl
- sudo apache2ctl configtest
- `sudo systemctl restart apache2`

Apache 2 Security

- **Step 3 — Configuring Apache Password Authentication**
- **Option 2: Configuring Access Control with .htaccess Files**
- **sudo nano /etc/apache2/apache2.conf**
- **<Directory /var/www/>**
- **Options Indexes FollowSymLinks**
- **AllowOverride All**
- **Require all granted**
- **</Directory>**

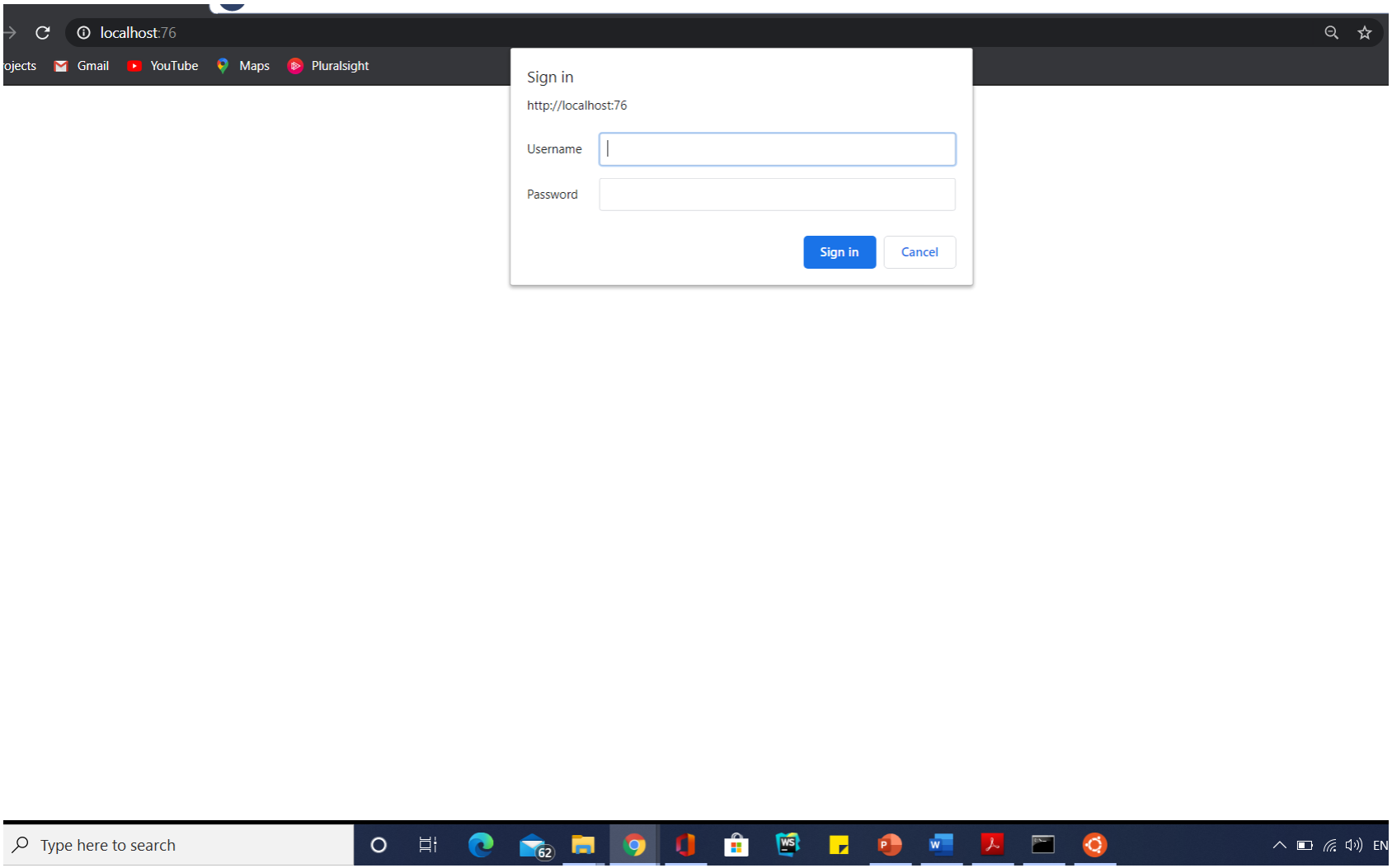
Apache 2 Security

- **Step 3 — Configuring Apache Password Authentication**
- **Option 2: Configuring Access Control with .htaccess Files**
- **sudo nano /var/www/html/.htaccess**
- **/var/www/html/.htaccess**
- **AuthType Basic**
- **AuthName "Restricted Content"**
- **AuthUserFile /etc/apache2/.htpasswd**
- **Require valid-user**

Apache 2 Security

- **Step 3 — Configuring Apache Password Authentication**
- **sudo systemctl restart apache2**
- **sudo systemctl status apache2**

Apache 2 Security



Questions

Module Summary

- Web Server Administration
- Apache virtual host workflow and configuration in detail
- Apache MPM configurations and usages
- Server capacity (like RAM, ROM) needed for n number of users (say n=3000+).
- Apache tuning when there is high server load
- Other Apache mechanism for handling uncertain situations