Apache for Linux - LPIC2

Franck Jeannot - 2016 - G185 - V1.0

177-202

hybrid format of 210 mm by 279.4 mm

CD568EA429B75E9EFE68F09682D7ED7B

1 Purpose

This document provides an **Apache** web server overview (Linux) with a specific focus on the objective **208.1: Basic Apache Configuration** of the LPI 117-202 Certification. Refer to [1]. A main source of this document is **"The LPIC-2 Exam Prep., 2014 Snow B.V"**

2 Acronyms and terminology

APXS : **AP**ache e**X**tenSion tool
DAC : **D**iscretionary **A**ccess **C**ontrol
DSO : **D**ynamic **S**hared **O**bjects
MAC : **M**andatory **A**ccess **C**ontrols

3 Apache

3.1 Objective 208.1: Basic Apache Configuration

Description: Candidates should be able to install and configure a web server. This objective includes monitoring the server's load and performance, restricting client user access, configuring support for scripting languages as modules and setting up client user authentication. Also included is configuring server options to restrict usage of resources. Candidates should be able to configure a web server to use virtual hosts and customise file access.

3.2 Key Knowledge Areas

- Apache 2.x configuration files, terms and utilities
- Apache log files configuration and content
- Access restriction methods and files
- mod_perl and PHP configuration
- Client user authentication files and utilities

[1] https://www.lpi.org/study-resources/lpic-2-202-exam-objectives/

- Configuration of maximum requests, minimum and maximim servers and clients
- Apache 2.x virtual host implementation with and without dedicated IP addresses)
- Using redirect statements in Apache's configuration files to customise file access

The following is a partial list of the used files, terms and utilities:

- access_log
- error_log
- .htaccess
- httpd.conf
- mod_auth
- htpasswd
- htgroup
- apache2ctl
- httpd

3.3 Installing the Apache web-server

Building Apache from source was routinely done when Apache emerged. Nowadays it is included in binary format in most distributions. Apache can be configured by setting options in configuration files. Where to find configuration files and how they are organized varies. Red Hat and similar distributions have their configuration files in the **/etc/httpd/conf/** directory. Other locations which are or have been used are **/etc/apache/config**, **/etc/httpd/config** and **/etc/apache2**. On most distributions the main configuration file is **httpd.conf**, in some cases it is **apache2.conf**.

3.4 Installing the Apache web-server with packages - OpenSuse-Leap-42.1

```
zypper in apache2 apache2-doc apache2-example-pages
#test the minimum configuration
apache2ctl configtest
#AH00557: httpd-prefork: apr_sockaddr_info_get() failed for ldsuse10
#AH00558: httpd-prefork: Could not reliably determine the server's fully qualified domain name,
    using 127.0.0.1. Set the 'ServerName' directive globally to suppress this message
#Syntax OK

Edit /etc/apache2/listen.conf to force the IP:port to listen
vi /etc/apache2/listen.conf
#Start the service
service apache2 start
#Check the process is listening correctly on port 80
netstat -tanlp |grep htt
#tcp 0 0 10.0.2.15:80 0.0.0.0:* LISTEN 22245/httpd-prefork
#Check the default page
w3m 10.0.2.15
#It works!

#restart at boot
chkconfig apache2 on
reboot
```

3.5 Installing the Apache web-server with packages - Debian/Kali/ubuntu

```
apt-get install apache2
/etc/init.d/apache2 start
/etc/init.d/apache2 stop
/etc/init.d/apache2 restart
```

An additional method to configure options for any subtree in the document tree is by including them in a **.htaccess** file. For security reasons you also need to enable the use of .htaccess files for any given subtree in the main configuration file, by setting

an AllowOverride directive for that Directory context. Any options set in such an .htaccess file influence the behaviour of the server for all files in that directory and in its subtrees, unless overridden by another .htaccess file or by directives in the main configuration file.
List of main configuration files (/etc/apache2 for OpenSuse example):

```
 #
-rw-r--r-- magic
-rw-r--r-- uid.conf
-rw-r--r-- ssl-global.conf
-rw-r--r-- server-tuning.conf
-rw-r--r-- mod_usertrack.conf
-rw-r--r-- mod_userdir.conf
-rw-r--r-- mod_status.conf
-rw-r--r-- mod_reqtimeout.conf
-rw-r--r-- mod_mime-defaults.conf
-rw-r--r-- mod_log_config.conf
-rw-r--r-- mod_info.conf
-rw-r--r-- mod_cgid-timeout.conf
-rw-r--r-- mod_autoindex-defaults.conf
-rw-r--r-- loadmodule.conf
-rw-r--r-- httpd.conf
-rw-r--r-- global.conf
-rw-r--r-- errors.conf
-rw-r--r-- default-server.conf
-rw-r--r-- charset.conv
lrwxrwxrwx mime.types -> ../mime.types
drwxr-xr-x vhosts.d
drwxr-xr-x ssl.prm
drwx------ ssl.key
drwxr-xr-x ssl.csr
drwxr-xr-x ssl.crt
drwxr-xr-x ssl.crl
drwxr-xr-x conf.d
drwxr-xr-x sysconfig.d
-rw-r--r-- listen.conf
```

List of main certificates files:

```
#ssl.crl   PEM-encoded X.509 Certificate Revocation Lists (CRL)
#ssl.crt   PEM-encoded X.509 Certificates
#ssl.csr   PEM-encoded X.509 Certificate Signing Requests
#ssl.key   PEM-encoded RSA Private Keys
#ssl.prm   public DSA Parameter Files
```

3.6 Modularity

Apache has a modular source code architecture. You can custom build the server using just the modules you really need. There are many modules available on the Internet. You can also write your own. Special modules exist that allow the use of interpreted languages like Perl and Tcl. They allow Apache to run interpreted scripts natively without having to reload an interpreter every time a script needs to be run e.g. **mod_perl** and **mod_tcl**.

3.7 APXS

APXS[2] is a tool for building and installing extension modules for the Apache HyperText Transfer Protocol (HTTP) server. This is achieved by building a dynamic shared object (DSO) from one or more source or object files which then can be loaded into the Apache server under runtime via the LoadModule directive from mod_so.

So to use this extension mechanism your platform has to support the DSO feature and your Apache httpd binary has to be built with the mod_so module. The apxs tool automatically complains if this is not the case. You can check this yourself by manually running the command:

```
#Install  APXS
zypper install apache2-devel

#Confirm that mod_so is compiled in your Apache version
httpd -l
#Compiled in modules:
#core.c
#mod_so.c
#http_core.c
#prefork.c
#mod_unixd.c
#mod_systemd.c
```

[2] https://httpd.apache.org/docs/current/programs/apxs.html

```
#Test APXS
cd /opt
wget https://github.com/omnigroup/Apache/archive/master.zip
mv master.zip apachemodules.zip
unzip apachemodules.zip
cd /opt/Apache-master/httpd/modules/ech

apxs -c -i -a mod_echo.c libapr-1.lib libaprutil-1.lib libapriconv-1.lib libhttpd.lib
#/usr/lib64/apr-1/build/libtool --silent --mode=compile gcc -std=gnu99 -prefer-pic -fmessage-
    length=0 -grecord-gcc-switches -O2 -Wall -D_FORTIFY_SOURCE=2 -fstack-protector -funwind-tables
     -fasynchronous-unwind-tables -g -fPIC -Wall -DLDAP_DEPRECATED  -DLINUX -D_REENTRANT -
    D_GNU_SOURCE -pthread -I/usr/include/apache2  -I/usr/include/apr-1   -I/usr/include/apr-1 -I/
    usr/include  -c -o mod_echo.lo mod_echo.c && touch mod_echo.slo
#/usr/lib64/apr-1/build/libtool --silent --mode=link gcc -std=gnu99    -o mod_echo.la  -rpath /usr
    /lib64/apache2 -module -avoid-version    mod_echo.lo libapr-1.lib libaprutil-1.lib libapriconv
    -1.lib libhttpd.lib
#/usr/share/apache2/build/instdso.sh SH_LIBTOOL='/usr/lib64/apr-1/build/libtool' mod_echo.la /usr/
    lib64/apache2
#/usr/lib64/apr-1/build/libtool --mode=install install mod_echo.la /usr/lib64/apache2/
#libtool: install: install .libs/mod_echo.so /usr/lib64/apache2/mod_echo.so
#libtool: install: install .libs/mod_echo.lai /usr/lib64/apache2/mod_echo.la
#libtool: finish: #PATH="/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/
    usr/bin/X11:/usr/games:/sbin" ldconfig -n /usr/lib64/apache2
#----------------------------
#Libraries have been installed in:
#/usr/lib64/apache2
#If you ever happen to want to link against installed libraries
#in a given directory, LIBDIR, you must either use libtool, and
#specify the full pathname of the library, or use the `-LLIBDIR'
#flag during linking and do at least one of the following:
#- add LIBDIR to the `LD_LIBRARY_PATH' environment variable
#during execution
#- add LIBDIR to the `LD_RUN_PATH' environment variable
#during linking
#- use the `-Wl,-rpath -Wl,LIBDIR' linker flag
#- have your system administrator add LIBDIR to `/etc/ld.so.conf'
See any operating system documentation about shared libraries for
```

```
more information, such as the ld(1) and ld.so(8) manual pages.
#----------------------------------
chmod 755 /usr/lib64/apache2/mod_echo.so
activating echo
```

We can find around 110 modules in /usr/lib64/apache2:

```
/usr/lib64/apache2>ls
mod_mime.so mod_ldap.so ... mod_ssl.so ... mod_setenvif.so mod_session_crypto.so
    mod_session_cookie.so mod_echo.so
```

3.8 Enhancing Apache performance

You can use the **MaxClients** setting to limit the amount of children your server may spawn hence reducing memory footprint.

3.9 httpd.conf

Extract from /etc/apache2/httpd.conf

```
# forbid access to the entire filesystem by default
<Directory />
Options None
AllowOverride None
<IfModule !mod_access_compat.c>
Require all denied
</IfModule>
<IfModule mod_access_compat.c>
Order deny,allow
Deny from all
</IfModule>
</Directory>
```

3.10 listen.conf

Extract from /etc/apache2/listen.conf

```
Listen 10.0.2.15:80
<IfDefine SSL>
<IfDefine !NOSSL>
<IfModule mod_ssl.c>
Listen 10.0.2.15:443
</IfModule>
</IfDefine>
</IfDefine>
```

3.11 access_log

The **access_log** contains a generic overview of page requests for your web-server. The format of the access log is highly configurable. The format is specified using a format string that looks much like a C-style printf format string. A typical configuration for the access log might look like the following:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

The format as shown is known as the Common Log Format (CLF). It is a standard format produced by many web servers and can be read by most log analysis programs. Log file entries produced in CLF will look similar to this line:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

This defines the nickname common and associates it with a particular log format string. The format as shown is known as the **Common Log Format** (CLF). It is a standard format produced by many web servers and can be read by most log analysis programs. Log file entries produced in CLF will look similar to this line:

```
127.0.0.1 - franck [dd/mm/yyyy:hh:mm:ss -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

CLF contains the following fields:
1. IP address of the client (%h)
2. RFC 1413 identity determined by identd (%l)
3. userid of person requesting (%u)
4. time server finished serving request (%t)
5. request line of user (%r)
6. status code servers sent to client (%s)
7. size of object returned (%b).

Extract from /var/log/apache2/access_log

```
/var/log/apache2> tail access_log
10.0.2.15 - - [05/Jul/2016:22:10:16 -0400] "GET /manual HTTP/1.1" 301 232 "-" "Mozilla/5.0 (X11;
    Linux x86_64; rv:47.0) Gecko/20100101 Firefox/47.0"
10.0.2.15 - - [05/Jul/2016:22:10:16 -0400] "GET /manual/ HTTP/1.1" 200 8964 "-" "Mozilla/5.0 (X11;
     Linux x86_64;
```

3.12 error_log

```
[Tue Jul 05 22:09:25.793896 2016] [ssl:warn] [pid 1919] AH01873: Init: Session Cache is not
   configured [hint: SSLSessionCache]
[Tue Jul 05 22:09:25.801335 2016] [mpm_prefork:notice] [pid 1919] AH00163: Apache/2.4.16 (Linux/
   SUSE) OpenSSL/1.0.1i-fips configured -- resuming normal operations
[Tue Jul 05 22:09:25.801451 2016] [core:notice] [pid 1919] AH00094: Command line: '/usr/sbin/httpd
   -prefork -f /etc/apache2/httpd.conf -D SYSCONFIG -C PidFile /var/run/httpd.pid -C Include /etc
   /apache2/sysconfig.d/ -D SYSTEMD -D FOREGROUND'
```

3.13 Restricting client user access

Many systems use either DAC or MAC to control access to objects: **discretionary access control (DAC)** A system that employs DAC allows users to to set object permissions themselves, they can change these at their discretion. **mandatory access controls (MAC)** A system that employs MAC has all its objects (files etc.) under strict control of a system administrator. Users are not allowed to set any permissions themselves. Apache takes a liberal stance and defines discretionary controls to be controls based on usernames and passwords, and mandatory controls to be based on static or quasi-static data like the IP address of the requesting client. Apache uses modules to authenticate and authorise users. In general modules will use some form of database to store and retrieve credential data. The mod_auth module for instance uses text files where mod_auth_dbm employs a DBM database. Below is a list of the security-related modules that are included as part of the standard Apache distribution. mod_auth (DAC) This is the basis for most Apache security modules; it uses ordinary text files for the authentication database. **mod_access (MAC)** This is the only module in the standard Apache distribution which applies what Apached defines as mandatory controls. It allows you to list hosts, domains, and/or IP addresses or networks that are permitted or denied access to documents.
mod_auth_anon (DAC) This module mimics the behaviour of anonymous FTP - rather than having a database of valid credentials, it recognizes a list of valid usernames (i.e., the way an FTP server recognizes "ftp" and "anonymous") and grants access to any of those with virtually any password. This module is more useful for logging access to resources and keeping robots out than it is for actual access control.

3.14 Configuring authentication modules

Apache security modules are configured by configuration directives. These are read from either the centralized configuration files (mostly found under or in the /etc directory) or from decentralized **.htaccess** files. The latter are mostly used to restrict access to directories and are placed in the top level directory of the tree they help to protect. For example, authentication modules will read the location of their databases using the **AuthUserFile** or **AuthDBMGroupFile** directives. This is an example of a configuration as it might occur in a centralized configuration file:

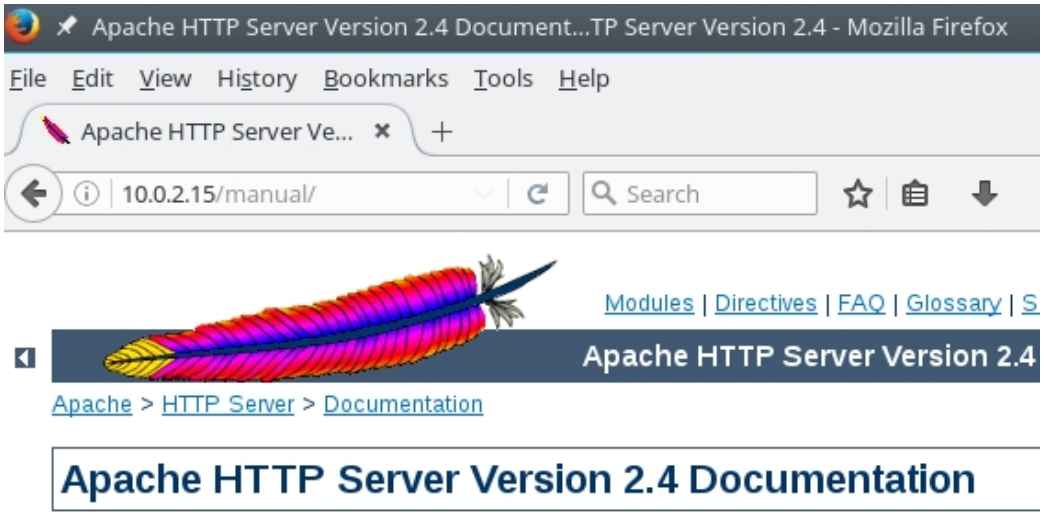
```
<Directory /home/johnson/public_html>
<Files foo.bar>
AuthName "Foo for Thought"
AuthType Basic
AuthUserFile /home/johnson/foo.htpasswd
Require valid-user
</Files>
</Directory>
```

The resource being protected is "any file named foo.bar" in the /home/johnson/public_html directory or any subdirectory thereof. Likewise, the file specifies whom are authorized to access foo.bar: any user that has credentials in the /home/johnson/-foo.htpasswd file. The alternate approach is to place a **.htaccess** file in the top level directory of any document tree that needs access protection. Note that you must set the directive **AllowOverride** in the central configuration to enable this. Example:

```
AuthUserFile {path to passwd file}
AuthGroupFile {path to group file}
AuthName {title for dialog box}
AuthType Basic
```

3.15 Review the local Apache documentation

Go to http://yourip/manual, if the apache2-doc package was correctly installed, all is available locally.



Refer to the **manual.conf** configuration file:

```
/etc/apache2/conf.d> cat manual.conf
# This configuration file belongs to the apache2-doc package.
# The alias provides the manual, even if you choose to move your DocumentRoot.
# Comment this out if you do not care for the documentation.
AliasMatch ^/manual(?:/(?:de|en|es|fr|ja|ko|ru))?(/.*)?$ "/usr/share/apache2/manual$1"
<Directory "/usr/share/apache2/manual">
Options Indexes
AllowOverride None
<IfModule !mod_access_compat.c>
Require all granted
</IfModule>
<IfModule mod_access_compat.c>
Order allow,deny
Allow from all
</IfModule>
```

```
<Files *.html>
SetHandler type-map
</Files>
SetEnvIf Request_URI ^/manual/(de|en|es|fr|ja|ko|ru)/ prefer-language=$1
RedirectMatch 301 ^/manual(?:/(de|en|es|fr|ja|ko|ru)){2,}(/.*)?$ /manual/$1$2
</Directory>
```

3.16 Review the local example pages

```
cat /srv/www/htdocs/index.html
<html><body><h1>It works!</h1></body></html>
```

3.17 Configuring mod_perl

mod_perl is another module for Apache, which loads the Perl interpreter into your Apache webserver, reducing spawning of child processes and hence memory footprint and need for processor power. Another benefit is code-caching: modules and scripts are loaded and compiled only once, and will be served from the cache for the rest of the server's life. Using **mod_perl** allows inclusion of Perl statements into your webpages, which will be executed dynamically if the page is requested. A very basic page might look like this:

```
print "Content-type: text/plain\r\n\r\n";
print "Hello, wold!\n";
```

3.18 apache2ctl

apachectl[3] is a front end to the Apache HyperText Transfer Protocol (HTTP) server. It is designed to help the administrator control the functioning of the Apache httpd daemon. The apachectl script can operate in two modes. First, it can act as a simple front-end to the **httpd** command that simply sets any necessary environment variables and then invokes httpd, passing through any command line arguments. Second, apachectl can act as a **SysV** init script, taking simple one-word arguments like start, restart, and stop, and translating them into appropriate signals to httpd. If your Apache installation uses non-standard paths,

[3] https://httpd.apache.org/docs/current/programs/apachectl.html

you will need to edit the apachectl script to set the appropriate paths to the httpd binary. You can also specify any necessary httpd command line arguments. See the comments in the script for details. A common configuration is /usr/sbin/apache2ctl -> apachectl

3.19 Apache Virtual Hosting

Virtual Hosting is a technique that provides the capability to host more than one domain on one physical host. There are two methods to implement virtual hosting:

- **Name-based virtual hosting** With name-based virtual hosting, the server relies on the client (e.g. the browser) to report the hostname as part of the HTTP headers. Using this technique, many different hosts can share the same IP address.
- **IP-based virtual hosting** Using this method, each (web) domain has it's own unique IP address. Since one physical host can have more than one IP address, one host can serve more than one (web)domain. In other words: IP-based virtual hosts use the IP address of the connection to determine the correct virtual host to serve. Note : Name-based virtual hosting eases the demand for scarce IP addresses. Therefore you should use name-based virtual hosting unless there is a specific reason to choose IP-based virtual hosting, see IP-based Virtual Hosting.

Name-based virtual hosting is a fairly simple technique. You need to configure your DNS server to map each hostname to the correct IP address first, then configure the Apache HTTP Server to recognize the different hostnames. To use name-based virtual hosting, you must designate the IP address (and possibly port) on the server that will be accepting requests for the hosts. This is configured using the NameVirtualHost directive.

```
### 'Main' server configuration ##############################################
#
# The directives in this section set up the values used by the 'main'
# server, which responds to any requests that aren't handled by a
# <VirtualHost> definition.  These values also provide defaults for
# any <VirtualHost> containers you may define later in the file.
#
# All of these directives may appear inside <VirtualHost> containers,
# in which case these default settings will be overridden for the
# virtual host being defined.
#
Include /etc/apache2/default-server.conf
```

```
### Virtual server configuration #############################################
#
# VirtualHost: If you want to maintain multiple domains/hostnames on your
# machine you can setup VirtualHost containers for them. Most configurations
# use only name-based virtual hosts so the server doesn't need to worry about
# IP addresses. This is indicated by the asterisks in the directives below.
#
# Please see the documentation at
# <URL:http://httpd.apache.org/docs/2.4/vhosts/>
# for further details before you try to setup virtual hosts.
#
# You may use the command line option '-S' to verify your virtual host
# configuration.
#
IncludeOptional /etc/apache2/vhosts.d/*.conf
```

The next step is to create a **<VirtualHost>** block for each different host you would like to serve. The argument to the <VirtualHost> directive should be the same as the argument to the **NameVirtualHost** directive (i.e., an IP address or * for all addresses). Inside each <VirtualHost> block you will need, at minimum, a **ServerName** directive to designate which host is served and a **DocumentRoot** directive to show where in the filesystem the content for that host lives. Suppose that both www.mydomain.com and www.otherdomain.net point to the IP address 10.10.10.12 . You then simply add the following to httpd.conf:

```
NameVirtualHost 10.10.10.12
<VirtualHost 10.10.10.12>
ServerName www.mydomain.com
DocumentRoot /www/domain
</VirtualHost>
<VirtualHost 10.10.10.12>
ServerName www.otherdomain.net
DocumentRoot /www/otherdomain
</VirtualHost>
```

3.20 Apache configuration cheat sheet

Hide apache version info

```
ServerSignature Off
ServerTokens Prod
```

Custom 404 Error message

```
ErrorDocument 404 /404.html
```

Create a virtual directory (mod_alias)

```
Alias /common /web/common
```

Permanent redirect (mod_alias)

```
Redirect permanent /old http://example.com/new
```

Create a cgi-bin

```
ScriptAlias /cgi-bin/ /web/cgi-bin/
```

Process .cgi scripts

```
AddHandler cgi-script .cgi
```

Add a directory index

```
DirectoryIndex index.cfm index.cfm
```

Turn off directory browsing

```
Options -Indexes
```

Turn on directory browsing

```
<Location /images>
Options +Indexes
</Location>
```

Create a new user for basic auth (command line)

```
htpasswd -c /etc/apacheusers
```

Apache basic authentication

```
AuthName "Authentication Required"
AuthType Basic
AuthUserFile /etc/apacheusers
Require valid-user
```

Only allow access from a specific IP

```
Order Deny,Allow
Deny from all
Allow from 127.0.0.1
```

Only allow access from your subnet

```
Order Deny,Allow
Deny from all
Allow from 176.16.0.0/16
```

mod_rewrite Turn on the rewrite engine

```
RewriteEngine On
```

Redirect /news/123 to /news.cfm?id=123

```
RewriteRule ^/news/([0-9]+)$ /news.cfm?id=$1 [PT,L]
```

Redirect www.example.com to example.com

```
RewriteCond %{HTTP_HOST} ^www\.example\.com$ [NC]
RewriteRule ^(.*)$ http://example.com$1 [R=301,L]
```