THE COMPREHENSIVE LAMP GUIDE

PHP, the "P" in LAMP, is a recursive acronym for PHP: Hypertext Preprocessor. It is the most widely used programming language for Web applications because of its ease of learning, implementation and wide range of server support. This guide aims to help you set up PHP on Apache for maximum performance.

PHP installation is similar to Apache's installation process, with GNU `autoconf` involved (`./configure` script). We've already discussed setting up the optimisation flags for Apache (`CFLAGS`, `CXXFLAGS`) To install PHP 5.3.6 (the latest available, as of this writing), you need (these requirements have been taken from the `INSTALL` file in the tarball):

- ANSI C Compiler (GCC on Linux)
- Flex 2.5.4
- Bison 1.28, 1.35 or 1.75
- Web server
- Module specific components (gd, pdf libs, etc)

The PHP source configuration is a bit tricky; extensions enabled by default (or by specifying an option) are compiled statically. You need to take special care about this. Statically compiling extensions means the extension will be embedded into the PHP binary itself; you cannot disable it at will. If you can't disable the

extension, it causes memory wastage if you aren't using the extension in any of your applications on the server.

Telling the PHP build system to build all extensions takes effort. You have to state "shared" in the extension enabler switch: `--with-EXT=shared` or `--enable-EXT=shared`. There are two switches, `--enable-shared` and `--disable-static` that can be passed to `./configure`, but they never worked for me — the build system always compiled extensions statically.
Get the PHP source tarball from its official website, and also note the MD5 sum shown on the download page, for file verification (similar to what we did when ~~installing MySQL, in July~~). Do not skip file verification; often, a proper tarball wasn't downloaded and my installation had problems like memory leaks, etc.
Extract the archive to get the `php-5.3.6` subdirectory. Before we begin configuring the source, if you want to install some PECL extensions directly with PHP (if you choose not to install PEAR/PECL with PHP), then you must place them in the `ext` directory, and subsequently run the`autoconf` tool present in the source code folder itself, with `./autoconf`).

Configuring and building PHP

So here's the table of options that the PHP `./configure` accepts, and a short description of what they do. This list isn't exhaustive; run `./configure --help | less` to get a full list. Please note that you have to set the hardware-dependent `CFLAGS`, `CXXFLAGS` for optimisation.
Another thing to remember is that if an option's *disable* version is shown in the exhaustive list, then its enable counterpart also exists and can be used. For example, `--disable-libxml` is listed, but not `--enable-libxml` — yet it can be used.

Option	Description
`cache-file=FILE`	Enables creation of `./configure` cache; `FILE` is usually`config.cache`. This improves configuration speed if the configuration process is broken due to the absence of some library, etc.
`--prefix=PREFIX`	Directory prefix where PHP should be installed, defaults to`/usr/local`; PHP binaries go in `<PREFIX>/bin`
`--with-apxs2=FILE`	Builds the Apache module `libphp5.so`. `FILE` is

Option	Description
	the path to the apxstool (this is optional; specify only when configure fails to find apxsin $PATH.
--disable-cli	Disables the command-line version of PHP and forces –-without-pear.
--enable-fpm	Enables building of FPM SAPI executable.
--disable-cgi	Disables CGI version of PHP; use if you won't be using CGI (for FastCGI, we have FPM).
--with-config-file-path=PREFIX/lib	Where the PHP interpreter should look for php.ini. I got confused with this — it's the path to the directory where php.ini will be, not including the filename! For example: --with-config-file-path=/usr/local/etc.
--with-config-file-scan-dir=PATH	Indicates which directory PHP scans to load additional configuration files. Helps clean up php.ini (separating it into different files). Also, if you install extensions, you don't have to add their configuration to php.ini; you can add separate configuration files to the scan directory. I use --with-config-file-scan-dir=/usr/local/etc/php.d
--enable/disable-libxml	Enables/disables libxml; it is enabled by default. You shouldn't disable this unless you have a special reason to do so, because many extensions depend on this.
--with-openssl[=shared[,DIR]]	Compile PHP with OpenSSL support. You can link the SSL extension with the OpenSSL library found in /usr/include, or a special version whose path you can provide as a parameter. If you want to build the extension as a dynamic library then: --with-openssl=shared
--with-zlib[=shared[,DIR]]	Enables zlib extension (responsible for gzip compression, etc.) Requires zlib >= 1.0.9.

Option	Description
--enable-bcmath[=shared]	Enables the bcmath extension, many packages use this.
--with-bz2[=shared[,DIR]]	Enables the bzip2 extension.
--enable-calendar[=shared]	Enables support for calendar conversion, some packages may require this.
--with-curl[=shared[,DIR]]	Enables the cURL extension. Requires the library be installed; many Web apps require this.
--enable-exif[=shared]	Enables the EXIF extension. A good idea if you have image-processing applications.
--enable-ftp[=shared]	Enables the FTP extension. There's no reason to enable this unless you are installing a PHP FTP client or want PHP scripts to connect to FTP servers.
--with-gd[=shared[,DIR]]	Enables GD support. Image-processing packages require this. GD is bundled with the source tarball, but you may configure the extension to use the system version of the gd library.
--with-jpeg-dir[=DIR]	Enables JPEG handling in GD. Requires libjpeg to be installed.
--with-png-dir[=DIR]	Enables PNG support in GD; needs libpng installed.
--with-xpm-dir[=DIR]	Enables XPM support in GD, needs libXpm installed.
--with-freetype-dir[=DIR]	Enables FreeType support in GD. FreeType is used by many packages, especially those that generate CAPTCHAs, etc.
--with-t1lib[=DIR]	Enables T1lib support, which is required by some packages, in GD.

Option	Description
--with-gettext[=shared[,DIR]]	Enables gettext support, used for internationalisation and localisation of programs. Some packages may require this.
--with-gmp[=shared,[,DIR]]	Enables GNU Math Processing library. This is similar to the BCMath extension. Most packages ask you to install either, but enabling both is better for sanity.
--with-mhash[=shared,[,DIR]]	Enables support for libmhash, which supports hash algorithms including common ones like MD5, SHA1, etc.
--enable-intl	Enables internationalisation support.
--enable-json[=shared]	JSON support is enabled by default; this option is not shown in the exhaustive list — yet it can be built as a shared extension.
--enable-mbstring[=shared]	Enables multi-byte string support, required for multilingual websites.
--with-mcrypt[=shared[,DIR]]	The mcrypt encryption library supports a lot of encryption algorithms.
--with-mysql[=shared,[,DIR]]	Enables MySQL support. DIR can be the path to the source files of the mysql library or mysqlnd; in the latter case, the native driver bundled in the source will be used. Defaults to /usr/local.
--with-mysql-sock[=DIR]	Sets the default location of the MySQL socket used by MySQL connect functions. If DIR is not specified, default locations are searched.
--with-mysqli[=shared[,FILE]]	Enables MySQLi support; this is an improved version of the MySQL extension with OOP interface. FILE is mysqlnd or path tomysql_config binary.
--enable-embedded-mysqli	Enables embedded MySQL server support for

Option	Description
	MySQLi; it doesn't work with mysqlnd.
--enable-pcntl[=shared]	Enables PCNTL (Process Control) extension, which is rarely required. This is enabled only for CLI and CGI (and FastCGI).
--enable-pdo[=shared]	PDO support is enabled by default, which is not shown in the exhaustive list.
--with-pdo-mysql[=shared[,DIR]]	Enables MySQL for PDO; DIR is mysqlnd or path to MySQL library.
--with-pdo-sqlite[=shared,DIR]]	SQLite 3 support for PDO, which is enabled by default; option not shown in exhaustive list. DIR is path to sqlite3 library which includes files.
--with-pspell[=shared[,DIR]]	Enables ASPELL spell-checker support; some Web applications use this. Needs GNU Aspell >= 0.5.0 installed on the system.
--enable-session[=shared]	Session support, which is enabled by default; option not shown in exhaustive list. There's no reason to build this extension as a shared library unless you have a special reason to do so.
--enable-shmop[=shared]	Enables SHM (shared memory) operation support.
--enable-simplexml[=shared]	Enabled by default; though not shown in exhaustive list.
--enable-soap[=shared]	Enables SOAP support, which some applications may need.
--with-sqlite[=shared[,DIR]]	SQLite 2 support, which is enabled by default, but not shown in exhaustive list.
--enable-sqlite-utf8	Enables UTF-8 support for SQLite 2
--with-sqlite3[=shared]	SQLite 3 support, which is enabled by default

Option	Description
	but not shown in exhaustive list.
--with-tidy[=shared]	The Tidy extension can clean up HTML markup to conform to W3C standards. You need tidy installed.
--with-xmlrpc[=shared[,DIR]]	XMLRPC-EPI support; some applications may use it, like blogs and CMSs.
--enable-xmlreader[=shared]	XMLReader support, which is enabled by default and is not shown in exhaustive list.
--enable-xmlwriter[=shared]	XMLWriter support, enabled by default but not shown in exhaustive list.
--enable-zip[=shared]	Enables ZIP-file handling extension.
--with-pear=DIR	Installs PEAR in DIR; DIR defaults to PREFIX/lib/php.

After you have completed configuration with ./configure and the options, run make and make install to install PHP, keeping your fingers crossed that it builds and installs successfully.

PHP configuration

Two configuration file candidates for php.ini are provided in the root of the source tarball:php.ini-production and php.ini-development. Unless you will be testing and developing on this system, choose the production version.
Copy php.ini-production to /usr/local/etc/php.ini or your /php.ini and start modifying it with your favourite text editor. Here is a table of options that can be specified in php.ini, along with a description of each:

Option	Description
short_open_tag=<On|Off>	The short open tag <? instead of <?php. If you run old applications, enable this; this may cause confusion if other language processors like XML, etc, are present.
output_buffering=<Off|Integer|On>	Maximum output data buffer size before sending to client. Onenables infinite buffer size (dangerous!). Best value: 4096.

Option	Description
zlib.output_compression=<Off|On>	Enables/disables gzip compression of output.
max_execution_time=<Integer>	Maximum script execution time (in seconds), set to 0 for CLI. Should be set after trial and error, though 30-60 seconds should be good for standard applications. A very large value is dangerous; a script can hog resources for a long time.
max_input_time=<Integer>	Maximum time a script can spend parsing request data. Default is unlimited (-1) [hard-coded for CLI].
memory_limit=<size>	Maximum memory a script may consume; size defaults to bytes, but modifiers like M, G can be applied, like 128 M. Keep it so that malicious scripts don't hog all available memory. 128-256 M is sufficient.
error_reporting	Sets the type of errors reported to stdout, stderr or the error log. Default value: E_ALL & ~E_NOTICE. Production value: E_ALL & ~E_DEPRECATED. Development value: E_ALL | ~E_STRICT
display_errors=<Off|On|stderr>	Displays errors to stdout or stderr. Stderr affects only CLI and CGI binaries.
include_path=<paths separated by colon>	Colon-separated paths for PHP to search for files named ininclude, require, include_once or require_once.
file_uploads = <Off|On>	Enables/disables file uploads.
upload_max_filesize=<size>	Maximum file-size for file upload; takes modifiers like M, G, etc.
max_file_uploads=<Integer>	Maximum number of files that can be uploaded in a single request.

Option	Description
allow_url_include=<Off|On>	Inclusion of PHP files from URLs. This can pose a security threat; malicious files can be included from remote servers.
extension_dir=<path>	Location to find PHP extensions.
extension=<filename>	Tells PHP to load the extension named. filename takes full path to the extension, else it will be sought in extension_dir.

In addition to this configuration, to get maximum performance, you should install an opcode cache like APC, XCache, eAccelerator, etc. Remember, PHP is an interpreted language; source is compiled every time the script is run. Caching compiled code saves CPU cycles.

You can also use Facebook's Hiphop to convert PHP code to C++, which will help you boost performance immensely — but that is beyond the scope of this article.

Configuring PHP with Apache

There are three methods to configure PHP on Apache: CGI (the worst option), mod_php (better), and mod_fastcgi (the best). I'll discuss only mod_php and mod_fastcgi methods.

The mod_php method

This is the most commonly used method to configure PHP with Apache. It is applicable only if you built the Apache SAPI for PHP (--with-apxs2). Add these lines to httpd.conf to enable PHP support with mod_php:

```
LoadModule php_module modules/libphp5.so
AddHandler php5-script .php
AddType text/html .php
```

If you have read the PHP documentation for installation, then you might know that PHP recommends the use of application/x-httpd-php for PHP scripts — but that never worked properly for me; hence, I won't advise it.

The mod_fastcgi method

With PHP-FPM, FastCGI usage has been increasing ever since, because of its advantages over CGI and mod_php. FastCGI's advantage over others is that the

PHP processing stack is separated from the server — there are some processes running separately, independent of the Web server on the machine where PHP is being used itself, or some remote destination. Because of this, opcode caches are able to share data across multiple processes, and their data is not destroyed when you change the Web server configuration and restart (or reload) it. Also, you can have dedicated PHP processing machines on the network to enable load sharing — very useful if you have a heavily trafficked site.
Again, there are two methods to use PHP with mod_fastcgi; the older uses spawn-fcgi or something similar, which sets up a PHP interpreter stack on a TCP port or a UNIX socket. The newer method uses PHP-FPM. mod_fastcgi is not provided with the default Apache installation — you have to download and install as per the INSTALL file in the tarball. Sometimes, the module is not automatically installed to the Apache modules directory; you need to copy it there from/.libs/.
After you have installed the module, to enable it in Apache, add (or uncomment) this directive:

```
LoadModule fastcgi_module modules/mod_fastcgi.so
```

The older method using spawn-fcgi

Run the following spawn-fcgi command (spawn-fcgi is a part of the lighttpd project, but is available as a separate package in many distributions):

```
spawn-fcgi -f /usr/local/bin/php-cgi -s /tmp/php.sock -u apache -g apache -C 10
```

This will launch the PHP interpreter stack (consisting of 10 processes and one manager process) that will listen for requests at /tmp/php.sock. You can also make the stack listen on a TCP port using -p (port) and -a (address) option instead of -s. These are mutually exclusive.
Add these lines to the Apache configuration to enable PHP:

```
AddHandler php-fcgi .php
FastCgiExternalServer /var/www/cgi-bin/php.external -socket /tmp/php.sock -pass-header Authorisation
Action php-cgi /cgi-bin/php.external
```

The above lines are the same if you use FPM — it is just that you may have to change the socket path if you specify a different path in FPM configuration.

You need to configure FastCgiExternalServer's path into a cgi-bin directory, or you may have to set the ExecCGI option for PHP scripts if php.external is not in a cgi-bin. There are various ways to set up php-fastcgi on Apache, so use the one that works for you. Use Google to search for more information.

FPM configuration

A sample FPM configuration may be placed in /usr/local/etc or <PREFIX>/etc. The default location seems to be /etc/php-fpm.conf (as per my installation, on Gentoo), but that may differ across distributions. Look for the sample configuration in /etc and /usr/local/etc or<PREFIX>/etc; copy it to php-fpm.conf in the same directory, and start modifying it. The configuration file is well documented, so I'll describe options specific to the PHP pool required to get PHP running, not the others.
A pool configuration section starts with the pool name in square brackets:

[www] ; Pool name = www
To make a pool listen on a socket or TCP port, you need to use the listen option:

listen = <path-to-socket|address:port>
Also note the options listen.owner, listen.group and listen.mode. If you make a mistake while configuring these, you may have a setup that doesn't work. These should be configured so that the user and group Apache is running under should be able to read and write to the socket. The options user and group in the FPM configuration specify the user and group PHP runs as. This means that the user and group specified there should have read-write permissions to the directories/files they may be processing.
Other options you need to configure
are pm, pm.max_children, pm.start_servers,pm.min_spare_servers and pm.max_spare_servers.
The pm=dynamic setting is best — it will launch PHP processes when required (of course, min_spare_servers number of processes will always be running).
Other options depend on your server's capacity. Read the documentation provided in the file itself to configure them.

Source : http://www.opensourceforu.com/2011/09/comprehensive-lamp-guide-part-3-php/