

CHAPTER 1



Installation and Configuration

It has been about 10 years since *PHP 5 Recipes – A Problem-Solution Approach* was published and PHP, and web technology in general, has come a long way since then. It's time to update some of the recipes.

The example code in this book is designed to work with the current versions (PHP 5.6 and PHP 7) but in many cases the examples will work with older versions of PHP 5. PHP 7 is still fairly new and many site owners have not yet upgraded. For the most part PHP 7 is compatible with PHP 5.6 but it does include a few new features and a couple of backwards compatibility issues. Not all extensions have been updated to work with the new Zend Engine (the internal workings of PHP).

It is highly recommended to use the very latest version of PHP. This will ensure the best performance and security, and it will provide access to the latest features.

PHP was designed as a scripting language for generating dynamic web content in the form of responses to requests from a client (usually a browser) to a web server using the HTTP protocol. The browser is no longer the only type of client for these scripts. In a world where more and more data is accessed through web services that return JSON, XML, or other structured formats, we are seeing clients being anything: from applications on a smart phone, other applications, and even servers talking to other servers to exchange data.

Recipe 1-1. Installing PHP

Problem

PHP is an open source scripting language that has been ported to many different platforms. It can run on anything from a Raspberry PI to a big IBM mainframe. Installing PHP used to require the user to compile the interpreter from source code, but today many operating systems provide packages that make it easy to install. No matter which operating system and web server you choose to use, the problem is the same: how do I get the web server to execute a script and return the response?

Solution

The number of supported operating systems, web servers, and server API's is quite large and the number of different combinations is even bigger. Most of the web servers on the Internet that use PHP as a scripting language are running some form of Linux and some are running Windows. When it comes to development environments there are Windows, Mac OSX, and Linux systems that are common. In addition to this there are at least three types of web servers: Apache, Nginx, and Internet Information Server (IIS) on Windows.

Electronic supplementary material The online version of this chapter ([doi:10.1007/978-1-4842-0605-8_1](https://doi.org/10.1007/978-1-4842-0605-8_1)) contains supplementary material, which is available to authorized users.

The common problem that all these configurations is solving is the injecting of a script parser into the web server so requests for certain files will result in invoking of a script and returning the output generated by the script. Other types of requests to the web server such as static HTML, images, JavaScript, and CSS files, etc., are handled by the web server directly as the server will resolve the request to a file on the file system and return the file unchanged to the requester.

The usual configuration is to tell the web server that files with a certain file extension (.php) are to be interpreted by PHP. When a request comes in for such a file the web server will pass the request to PHP and wait for the output.

How it Works

With the large number of possible combinations of operating systems, web servers, and ways to link PHP to the web server, I have chosen a few combinations that are described in the following few examples. The first example covers how to install Apache and PHP on a clean CentOS 7 operating system using the package management tool yum. A later example will show how to do the same on Windows.

The package manager yum is used to install, update, or remove packages from the CentOS repository. It can also be used to list the available packages. It is always a good idea to make sure the system is fully up to date before installing new packages. This is done by running the following command:

```
yum update
```

This will ensure all kernel and application packages are updated to the latest version. After installing a new kernel the system must be rebooted.

The name of the apache package is called httpd, so to list all the packages related to the Apache web server we could issue the following command:

```
# yum list httpd*
Available Packages
httpd.x86_64                2.4.6-40.el7.centos          base
httpd-devel.x86_64         2.4.6-40.el7.centos          base
httpd-manual.noarch        2.4.6-40.el7.centos          base
httpd-tools.x86_64         2.4.6-40.el7.centos          base
```

Note that many of the yum commands should be run as the root user. Either use su - to switch to the root account or use sudo in front of each command to execute that command as the root user.

The output shows that there are five packages available in this environment. It is only necessary to install the first package. It can be done with the command.

```
yum install httpd
```

This will install all packages needed to get the basic web server installed. The Apache web server depends on the packages apr, apr-util, httpd-tools, and mailcap. After installation the server can be configured to start automatically by calling.

```
systemctl enable httpd
```

And started with this command:

```
systemctl start httpd
```

This will provide you with a standard Apache 2.4 configuration. It is ready to serve static html files, but there are still a few steps to do before it can serve requests to PHP scripts. To test the new web server configuration it is necessary to create the first HTML document. The default configuration places the document root at `/var/www/html`. In addition the default configuration is set to default to a document called `index.html`. This is used if the request is made without a specific document. Use VI or your preferred editor to create `index.html` in the folder `/var/www/html`. The content could look like this:

```
<!DOCTYPE html>
<html>
<head>
  <title>Apache</title>
</head>
<body>
  <h1>It works!</h1>
</body>
</html>
```

Testing of the web server as responding can be done with the `wget` command or by pointing a web browser to the host name or IP address of the server. Depending on the configuration of the operating system you might have to install the `wget` command first. If you have a firewall installed you might also have to configure that to allow tcp traffic on port 80 to the server.

Instead of a static HTML document we can now create the first simple PHP script. This will not do much until PHP is installed, but it will showcase how PHP scripts work on a server that's not configured to handle requests for PHP scripts. The very first PHP script could be a simple one like this:

```
<?php
// 1_1.php
phpinfo();
?>
```

The last closing tag `?>` is not necessary unless the file contains any content after the closing tag that is going to be sent directly to the client as is, without PHP interpreting the content. In fact it is a common standard to exclude the closing `?>` on scripts that only contain PHP code.

If the browser is pointed to the web server (in my case the address is <http://192.168.23.148/phpinfo.php>), the output will show the content of the file.

```
<?php
phpinfo();
?>
```

The next step is to install one or more PHP packages. To get a list of the available packages use the `yum` command:

```
# yum list PHP*
Available Packages
php.x86_64                    5.4.16-36.el7_1      base
php-bcmath.x86_64           5.4.16-36.el7_1      base
php-cli.x86_64               5.4.16-36.el7_1      base
php-common.x86_64           5.4.16-36.el7_1      base
php-dba.x86_64               5.4.16-36.el7_1      base
```

php-devel.x86_64	5.4.16-36.el7_1	base
php-embedded.x86_64	5.4.16-36.el7_1	base
php-enchant.x86_64	5.4.16-36.el7_1	base
php-fpm.x86_64	5.4.16-36.el7_1	base
php-gd.x86_64	5.4.16-36.el7_1	base
php-intl.x86_64	5.4.16-36.el7_1	base
php-ldap.x86_64	5.4.16-36.el7_1	base
php-mbstring.x86_64	5.4.16-36.el7_1	base
php-mysql.x86_64	5.4.16-36.el7_1	base
php-mysqlnd.x86_64	5.4.16-36.el7_1	base
php-odbc.x86_64	5.4.16-36.el7_1	base
php-pdo.x86_64	5.4.16-36.el7_1	base
php-pear.noarch	1:1.9.4-21.el7	base
php-pecl-memcache.x86_64	3.0.8-4.el7	base
php-pgsql.x86_64	5.4.16-36.el7_1	base
php-process.x86_64	5.4.16-36.el7_1	base
php-pspell.x86_64	5.4.16-36.el7_1	base
php-recode.x86_64	5.4.16-36.el7_1	base
php-snmp.x86_64	5.4.16-36.el7_1	base
php-soap.x86_64	5.4.16-36.el7_1	base
php-xml.x86_64	5.4.16-36.el7_1	base
php-xmlrpc.x86_64	5.4.16-36.el7_1	base

Note that the version is 5.4.16 although the custom version is 5.6.20 or 7.0.5. This is the case with many Linux distributions. They do backport security updates but they might not backport new features or new versions. It is possible to get PHP from other repositories other than the official CentOS one, but if a newer version is needed, I recommend compiling it from source as described in Recipe 1-3.

The basic package is all that's needed, but installing the `php-cli`, `php-common`, and one or more database packages is most likely going to work for most people. The packages can be installed one by one or you can choose to install all the packages. The first example shows how to install a few packages.

```
yum install php php-cli php-common php-mysql
```


This will also install `libzip` and `php-pdo`.

Installing all packages is done with `yum install php*`.

After installation of the PHP packages it is necessary to restart the Apache server. This will make sure the installed Apache module is loaded.

```
systemctl restart httpd
```


Pointing the browser to the web server (<http://192.168.23.148/phpinfo.php> or <http://localhost/phpinfo.php>) will now yield this output:

PHP Version 5.4.16


System	Linux localhost.localdomain 3.10.0-327.13.1.el7.x86_64 #1 SMP Thu Mar 31 16:04:38 UTC 2016 x86_64
Build Date	Jun 23 2015 21:18:22
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/curl.ini, /etc/php.d/fileinfo.ini, /etc/php.d/json.ini, /etc/php.d/mysql.ini, /etc/php.d/mysqli.ini, /etc/php.d/pdo.ini, /etc/php.d/pdo_mysql.ini, /etc/php.d/pdo_sqlite.ini, /etc/php.d/phar.ini, /etc/php.d/sqlite3.ini, /etc/php.d/zip.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API220100525,NTS
PHP Extension Build	API20100525,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies

Powered By



The actual output will show much more data but this shows that PHP is now working on the server.

It is not that common to have Linux installed on a desktop or laptop computer. If you want to be able to test the PHP code locally before deploying it to a server you will need a local web server that supports PHP. It is also possible to do all the development remotely, but that will require Internet access.

Use the following commands to show the actual version of Apache and PHP that's installed on the system:

```
# httpd -v
Server version: Apache/2.4.6 (CentOS)
Server built:   Nov 19 2015 21:43:13
```

And for PHP

```
# php -v
PHP 5.4.16 (cli) (built: Jun 23 2015 21:17:27)
Copyright (c) 1997-2013 The PHP Group
Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies
```

And to get a list of the installed PHP extensions use the `php -m` command:

```
# php -m
[PHP Modules]
bz2
calendar
Core
ctype
curl
date
ereg
exif
fileinfo
filter
ftp
gettext
gmp
hash
iconv
json
libxml
mhash
mysql
mysqli
openssl
pcntl
pcre
PDO
pdo_mysql
pdo_sqlite
Phar
readline
Reflection
```

```

session
shmop
SimpleXML
sockets
SPL
sqlite3
standard
tokenizer
xml
zip
zlib

```

[Zend Modules]

On the Mac OSX platform, PHP comes preinstalled. On my version of Mac OSX (El Capitan) the installed versions can be listed with the same commands as above:

```

$ httpd -v
Server version: Apache/2.4.16 (Unix)
Server built:   Jul 31 2015 15:53:26
$ php -v
PHP 5.5.30 (cli) (built: Oct 23 2015 17:21:45)
Copyright (c) 1997-2015 The PHP Group
Zend Engine v2.5.0, Copyright (c) 1998-2015 Zend Technologies

```

The default-installed modules are the following:

```

$ php -m
[PHP Modules]
bcmath
bz2
calendar
Core
ctype
curl
date
dba
dom
ereg
exif
fileinfo
filter
ftp
gd
hash
iconv
json
ldap
libxml
mbstring
mysql

```

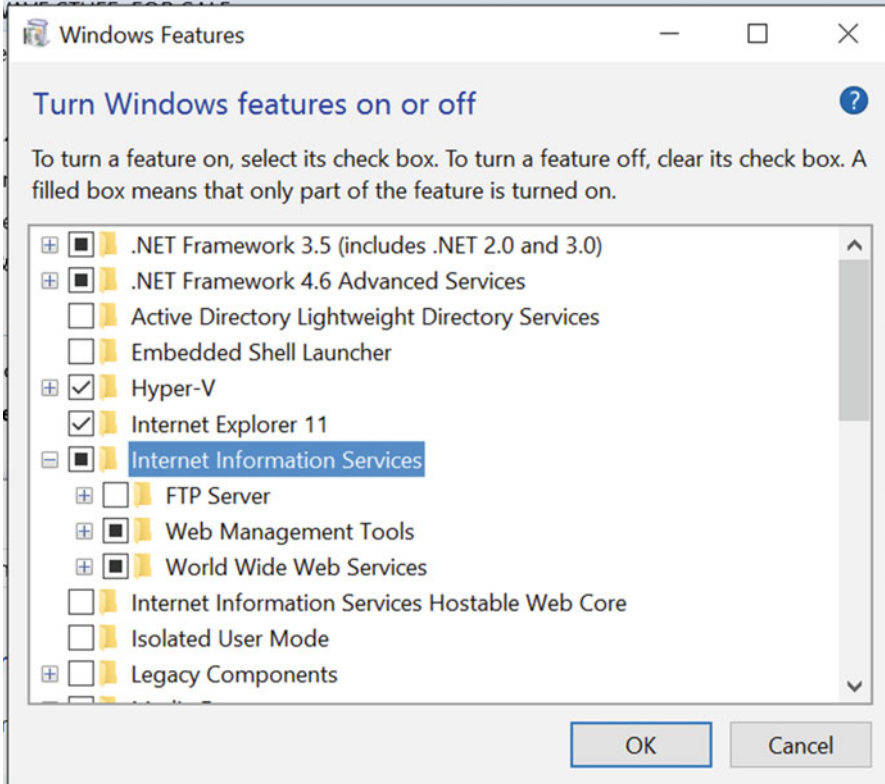
mysqli
mysqlnd
openssl
pcre
PDO
pdo_mysql
pdo_sqlite
Phar
posix
readline
Reflection
session
shmop
SimpleXML
snmp
soap
sockets
SPL
sqlite3
standard
sysvmsg
sysvsem
sysvshm
tidy
tokenizer
wddx
xml
xmlreader
xmlrpc
xmlwriter
xsl
zip
zlib

[Zend Modules]

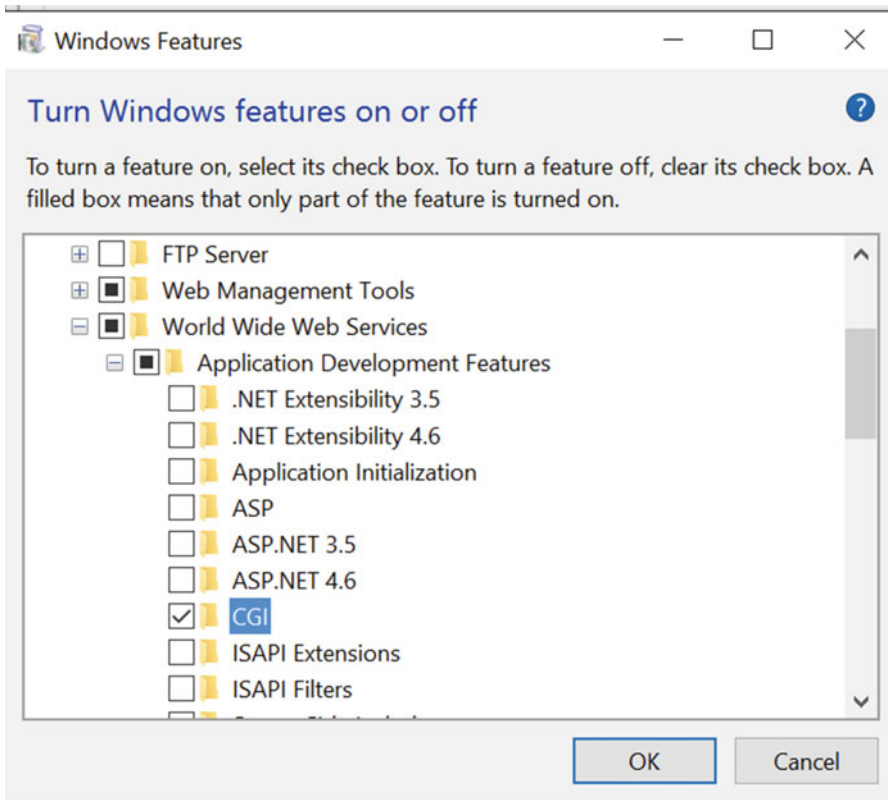
Getting PHP to run on Windows is almost as easy. Please note that PHP no longer supports older versions of Windows (XP/2003 etc.). The first step is to install a web server: IIS or Apache. These days there is not much difference in performance or features between the two. If you are going to deploy the PHP scripts on IIS on a Windows server it's recommended to do the development on a similar platform. The same goes for Apache deployments.

When developing on Windows and deploying on Linux it is important to note a few key differences between the two operating systems. First of all, file names on Windows are case insensitive. If your script refers to a file called MyFile.php but the actual file name is myfile.php, it will work on Windows but break when you deploy to the Linux platform. Secondly there is the matter of directory separator. This is a slash (/) on Linux and Unix environments and a backslash (\) on windows. PHP can use both when referencing local files but always use slashes in URL's. It is recommended to always use the slash as a directory separator on Windows because there is no need to escape that character when it's written as part of a double quoted string.

To install Internet Information Services (IIS version 7 or later) on a Windows 10 system, launch the control panel and click on “Programs and Features.” In the left bar there is a link to “Turn Windows features on or off.” This will bring up a window with Windows Features. Simply enable the option for Internet Information Services and click on the Ok button, as shown in the image below.



Make sure to expand the World Wide Web Services folder, then expand Application Development Features and select the CGI option.

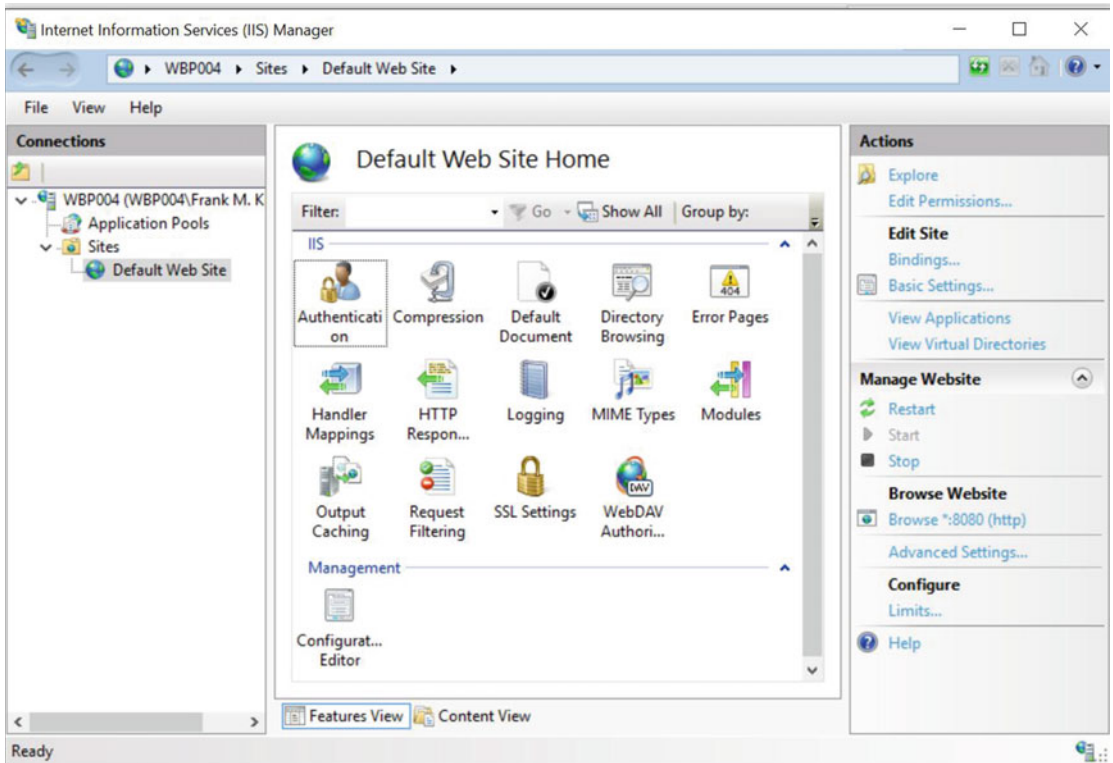


This will install the web server, the CGI module needed for PHP, and the management tools. Be aware that other processes (like Skype and Plex) installed on the computer might already be using the default web port (80). If that is the case you can either disable these services or use a different port for the web server. Using different ports for each web site you are working on will allow you to host multiple sites. It will also allow concurrent use of Apache and IIS on the same system.

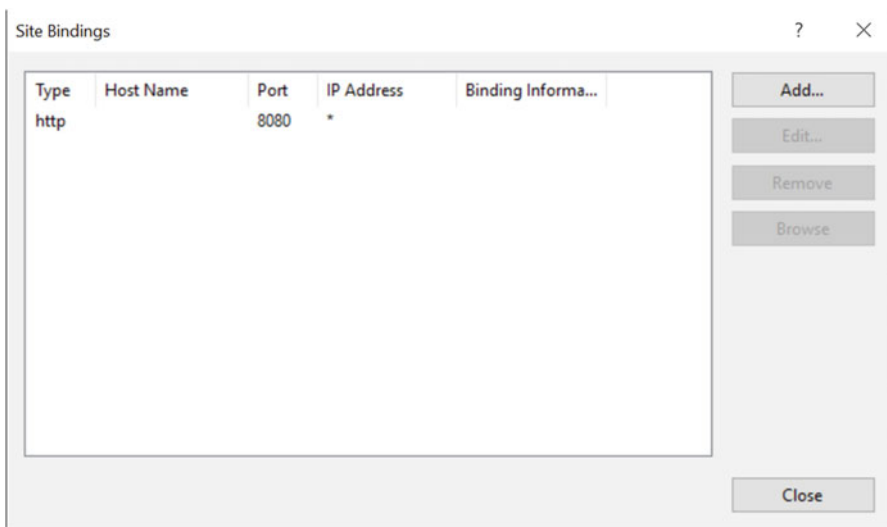
It is possible to have multiple web sites (on the same server) share a single port. Using different host names for each web site does that. These host names can be managed locally in a hosts file. This file is located in `c:\Windows\System32\Drivers\etc` on a Windows system and in `/etc` on a Mac or Linux system.

The default IIS configuration creates a web server that listens on port 80 and uses `%SystemDrive%\inetpub\wwwroot` as the document root. `%SystemDrive%` normally resolves to `C:\`. It is not necessary to use this directory structure for the web site.

The next step is to start the web site. In my case I had to use a different port as port 80 is used for Skype. Use the IIS manager application to make changes to the site. The Internet Information Services (IIS) manager can be found in the control panel under administrative tools or by typing IIS in the search bar. The application looks like this:



Right-click on the Default Web Site and select the Edit Bindings option and double-click on the line to change the configuration. Only the port number is changed from 80 to 8080. It's also possible to limit the IP address or host name, and it's possible to add additional bindings to bind the same web site to different port numbers and/or host names if needed.



The final step is to start the site. Right-clicking on the Default Web Site in the left pane and then the Manage Website option to expand the menu and select the Start option does this.

The web site can now be accessed from a browser. In this case the URL is `http://localhost:8080`. As with the Apache installation on Linux this site only serves static HTML and ASP documents if ASP is enabled. There are some additional steps needed to get support for PHP.

This starts by downloading the PHP binary package from <http://windows.php.net/download>. There are a number of different versions to choose from. It is recommended to use the latest version, compiled with the latest version of Microsoft Visual Studio. That is currently PHP 7.0.5 compiled with Visual Studio 2015 (VC14). Simply download the zip file and expand it into `C:\PHP`, or any other folder. This will give a directory structure that looks like this:

```
PS C:\PHP> ls
```

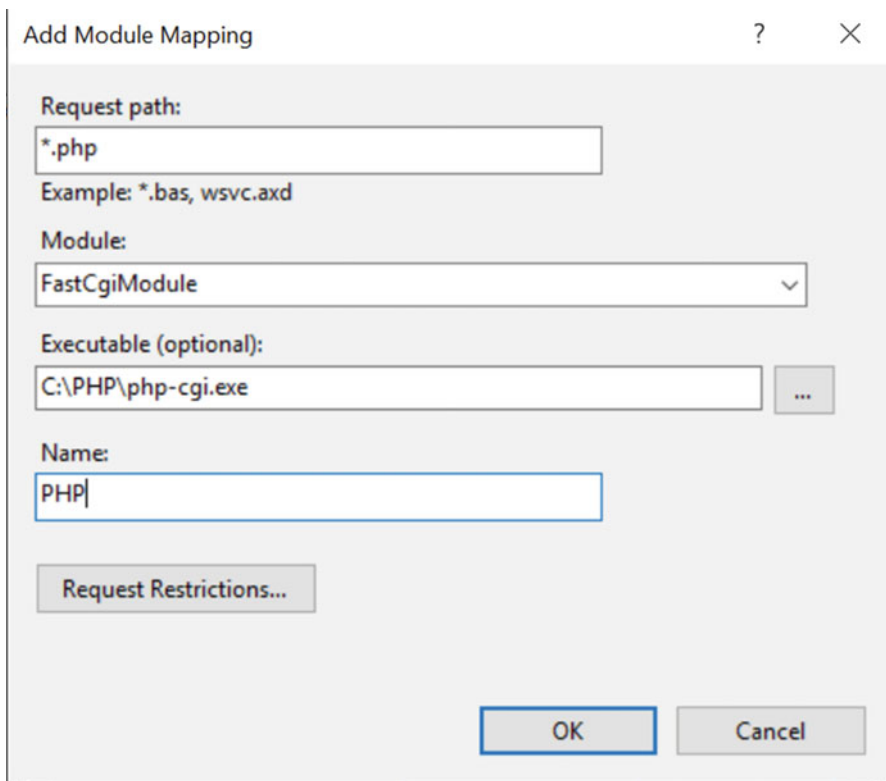
```
Directory: C:\PHP
```

Mode	LastWriteTime	Length	Name
d-----	4/2/2016	13:48	dev
d-----	4/2/2016	13:48	ext
d-----	2/13/2016	14:52	extras
d-----	2/13/2016	14:52	lib
d-----	4/2/2016	13:48	sasl2
-a----	4/2/2016	13:49	98816 deplister.exe
-a----	4/2/2016	13:49	1175552 glib-2.dll
-a----	4/2/2016	13:49	15872 gmodule-2.dll
-a----	4/2/2016	13:49	25048064 icudt56.dll
-a----	4/2/2016	13:49	1820160 icuin56.dll
-a----	4/2/2016	13:49	41984 icuio56.dll
-a----	4/2/2016	13:49	226816 icule56.dll
-a----	4/2/2016	13:49	1188864 icuuc56.dll
-a----	4/2/2016	13:49	79407 install.txt
-a----	4/2/2016	13:49	1389568 libeay32.dll
-a----	4/2/2016	13:49	36352 libevent.dll
-a----	4/2/2016	13:49	135168 libpq.dll
-a----	4/2/2016	13:49	77824 libsasl.dll
-a----	4/2/2016	13:49	235008 libssh2.dll
-a----	4/2/2016	13:49	3286 license.txt
-a----	4/2/2016	13:49	51511 news.txt
-a----	4/2/2016	13:49	43 phar.phar.bat
-a----	4/2/2016	13:49	53242 pharcommand.phar
-a----	4/2/2016	13:49	52736 php-cgi.exe
-a----	4/2/2016	13:49	30720 php-win.exe
-a----	4/2/2016	13:49	98304 php.exe
-a----	4/2/2016	13:49	2523 php.gif
-a----	4/2/2016	13:49	70752 php.ini-development
-a----	4/2/2016	13:49	70784 php.ini-production
-a----	4/2/2016	13:49	7088640 php7.dll
-a----	4/2/2016	13:49	869002 php7embed.lib
-a----	4/2/2016	13:49	197632 php7phpdbg.dll
-a----	4/2/2016	13:49	214528 phpdbg.exe
-a----	4/2/2016	13:49	20183 readme-redis-bins.txt
-a----	4/2/2016	13:49	12722 snapshot.txt
-a----	4/2/2016	13:49	274944 sslay32.dll

In most cases it's also necessary to install the redistribution libraries provided by Microsoft. It is important to install the right version of these but you can install them all and some of them might be installed already from other programs and tools installed on the system. The links to the different versions can be found in the left pane on the site <http://windows.php.net>.


The final step is to enable PHP in the web server. This is similar to how it was done for Apache on Linux. IIS uses the FastCGI interface to communicate between IIS and PHP. On Linux, PHP was installed and loaded as a module under the web server.

Start by clicking on the Default Web Site in the left pane of the IIS Manager. That will bring up a list of icons. Then double-click on the icon called Handler Mappings. Then click on the Add Module Mapping link in the right pane. This opens an input dialog with four input boxes. The first one is where the request path is defined. We want the PHP interpreter to be invoked on all files ending in .php so the value should be *.php. The next box is a drop-down with a number of different modules to choose from. Make sure to select the FastCGIModule and not the CGIModule. The interface between the web server and the module are different and the request will fail if the wrong version is used. In the third box enter the path to php-cgi.exe (or use the browse button to locate the file) and finally enter a name for the Module Mapping. I chose PHP as shown in the screen shot below.




The IIS Web server is now configured to serve requests to PHP scripts. Restart the web server and place the same phpinfo.php file we used before in the document root (c:\inetpub\wwwroot). If you want to access the server from other computers you will have to turn the firewall off or add a rule for the TCP port 8080 (if that's the one you used).

The sample script will generate output like this:

PHP Version 7.0.5


System	Windows NT WBP004 10.0 build 10586 (Windows 10) i586
Build Date	Mar 30 2016 09:57:56
Compiler	MSVC14 (Visual C++ 2015)
Architecture	x86
Configure Command	cscrip /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--disable-zts" "--with-pdo-oci=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgsql"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	(none)
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS,VC14
PHP Extension Build	API20151012,NTS,VC14
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, phar
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, mcrypt.*, mdecrypt.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies



Apache can be installed with binaries provided by Apache Lounge. The links to these can be found on <http://windows.php.net>. Download the zip file with the version that matches your system and unzip the contents to C:\Apache24. This directory matches the configuration in the httpd.conf file. If you choose to install in a different directory, update the httpd.conf file to match the location.

The default configuration is also using port 80. In order to run both IIS and Apache and possible other applications working on port 80 it is necessary to change the port for Apache. In this example I change the port to 8090 by updating the line in c:\Apache24\conf\httpd.conf

From

Listen 80

To

Listen 8090

The final step is to start the web server with this command:

```
C:\Apache24\bin\httpd.exe
```

This command is useful to create a single instance of the server. The server will only be running as long as this process is running. It is also possible to install a service that starts automatically when the computer is started. Adding `-k install` as an argument to the start command does this:

```
C:\Apache24\bin\httpd.exe -k install
```

As before, pointing a browser to the server, it's now possible to test the web server. In this case the URL is `http://localhost:8090`.

The final step is to configure the server to support PHP requests. This can be done by using the Apache module that comes with the thread safe version of PHP or by using the FastCGI interface used for the IIS server.

Configuring with the Apache module requires these changes:

Start by creating a file called `C:\Apache24\conf\extras\httpd-php.conf` and add the following lines:


```
#
LoadModule php7_module "C:/PHP/php7apache2_4.dll"
AddHandler application/x-httpd-php .php

<IfModule php7_module>
    PHPIniDir "C:/PHP"
</IfModule>
```

Then add the following line to the end of `C:\Apache24\conf\https.conf`


```
Include conf/extra/httpd-php.conf
```

Starting the Apache server will now support PHP requests and the output from `phpinfo()` looks like this:

PHP Version 7.0.5


System	Windows NT WBP004 10.0 build 10586 (Windows 10) AMD64
Build Date	Mar 30 2016 09:56:03
Compiler	MSVC14 (Visual C++ 2015)
Architecture	x64
Configure Command	<code>cscrip /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-sdk\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"</code>
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	(none)
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,TS,VC14
PHP Extension Build	API20151012,TS,VC14
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, phar
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, mcrypt.*, mdecrypt.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies



In order to configure Apache to use the FastCGI version of PHP it is necessary to download and install an additional Apache module called `mod_fcgid.so`. This file should be placed in `c:\Apache24\modules` and the configuration file for PHP (`C:\Apache24\conf\extras\httpd-php.conf`) should be changed to this:

```
#
LoadModule fcgid_module modules/mod_fcgid.so
FcgidInitialEnv PHPRC "C:/PHP"
AddHandler fcgid-script .php
FcgidWrapper "C:/PHP/php-cgi.exe" .php
```

The first line can also be placed in the main configuration file (`C:\Apache24\conf\httpd.conf`). The second line instructs PHP to look for the `php.ini` file in `C:\PHP`. The third line instructs Apache to handle requests for `.php` files via the FastCGI API, and the last line instructs FastCGI to call the `php-cgi.exe` executable.

In addition it is necessary to add an option called ExecCGI to allow Apache to execute the CGI script. This is either done in the main file by changing the Options in the <Directory> section from “Options Indexes FollowSymLinks” to “Options ExecCGI Indexes FollowSymLinks” or by a similar change for each virtual host that needs the same permissions.

If we load the phpinfo.php file in a browser the output will now show that the server is using the CGI/FastCGI server API instead of the Apache2Handler.

Using the FastCGI interface in Apache and IIS makes it possible to install multiple versions of PHP on the same system. Creating multiple virtual hosts pointing to the same document root but utilizing different CGI handlers makes it very easy to test the application code with different versions of PHP.

Recipe 1-2. Configuring PHP

Problem

The focus in the previous section was to install and perform basic configuration of a web server to be used with PHP. There are a large number of configuration options for PHP that works independently of the web server. What is the best approach to configuring PHP?

Solution

The basic configuration is done through a file called php.ini. It can be located in a number of different places depending on how PHP is compiled and installed and which operating system of PHP it is installed on. In addition it's possible to have different php.ini files based on which server API is used to execute the PHP script. This makes it possible to have one php.ini file that's used by the web server and a different file that is used when the CLI version of PHP is used to execute a script as a cron job or command line.

A common location of the php.ini file(s) is in /etc/php.ini on most Linux and Mac OSX systems, and on Windows it's common to place these files in the C:\Windows folder but it can be placed in other locations based on environment variables or other configuration options.

If php.ini is the only file that's found it will be used by all server API's. There is a special naming convention that makes it possible to have files specific to each server API. The name of each of these can be used as the replacement for ASPI in the name of the php-SAPI.ini file. Common names for SAPI's are cli, apache, apache2handler, cgi-fcgi, isapi, and many more.

Some of the settings in php.ini can be set or overwritten at runtime. This is done with the `ini_set()` function. Some settings have an impact on resources and security, and these can only be defined in the main php.ini file (system administrator), and others can be set in per directory bases and some can be set by the script at runtime (`ini_set()`).

It is common practice to use one configuration for the development environment and another for the production environment. In the development environment it's often helpful to get error and warnings shown directly in the web page, and in the production environment it's preferred not to show any errors or warnings that might expose vulnerabilities or simply confuse the visitor.

If the server is hosting multiple virtual hosts it can be necessary to overwrite certain values on the php.ini file for one or more of these virtual hosts. That way it's possible to have one set of error reporting for the development instance and a different set for the production instance even though both sites are installed on the same web server, sharing a single php.ini file. These overwrites can be defined in Apache configuration where the virtual host is defined or in a .htaccess file in a specific directory.

It is recommend to minimize the use of .htaccess files. These files are parsed on every request and this can have an impact on performance. Using the Apache configuration or a php.ini file will cause the settings to be read once, when the server is started.

If the `display_errors` option is set to `off` in `php.ini` and it is decided to set a different value for a single host it can be done by adding a `php_flag` to the `<VirtualHost></VirtualHost>` section of the Apache configuration. This can either be in the main `httpd.conf` file or in a file specific to the host as shown in the following sample.

```
<VirtualHost *:80>
  ServerName www.example.com
  DocumentRoot /var/www/example.com/root
  DirectoryIndex index.html index.php
  php_flag display_errors on
</VirtualHost>
```

The option `php_flag` is used for Boolean options. Other `php.ini` values can be set or overwritten with the `php_value` option. To set the value for `error_reporting` to `E_ALL` (showing all errors, warnings, and notices, etc.), you will have to use the integer value for `E_ALL`. When configured in the `php.ini` file it's possible to use the names (`E_ALL`, `E_ERROR`, `E_NOTICE` etc.). These values are not known within the Apache configuration and we have to use the corresponding integer value. Finding the value of the `E_ALL` constant can be done with the following command-line script:

```
# php -r 'echo E_ALL;'
```

Using the `-r` option when running the CLI version of PHP is an easy way to run a simple one-line PHP script. The script will show that the value is the sum of the 15 first bits of an integer or 32767. So to set the `error_reporting` to `E_ALL` in the Apache configuration use the following line:

```
php_value error_reporting 32767
```

How it Works

When PHP is installed from source it includes two different versions of the `php.ini` file called `php.ini-development` and `php.ini-production`. These two files can be used as a recommended baseline for the two different environments. In most cases these contain a good starting point for configuration of PHP, but in most cases and as new features are added to the web site it's necessary to add or change certain values. At the top of these files there is a Quick Reference section, a list of options that would normally be different in the two environments or different from the default value if no setting is provided:

```
;;;;;;;;;;;;;;;;;;;;;;;;;
; Quick Reference ;
;;;;;;;;;;;;;;;;;;;;;;;;;
; The following are all the settings that are different in either the production
; or development versions of the INIs with respect to PHP's default behavior.
; Please see the actual settings later in the document for more details as to why
; we recommend these changes in PHP's behavior.

; display_errors
;   Default Value: On
;   Development Value: On
;   Production Value: Off
```

```
; display_startup_errors
;   Default Value: Off
;   Development Value: On
;   Production Value: Off

; error_reporting
;   Default Value: E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED
;   Development Value: E_ALL
;   Production Value: E_ALL & ~E_DEPRECATED & ~E_STRICT

; html_errors
;   Default Value: On
;   Development Value: On
;   Production value: On

; log_errors
;   Default Value: Off
;   Development Value: On
;   Production Value: On

; max_input_time
;   Default Value: -1 (Unlimited)
;   Development Value: 60 (60 seconds)
;   Production Value: 60 (60 seconds)

; output_buffering
;   Default Value: Off
;   Development Value: 4096
;   Production Value: 4096

; register_argc_argv
;   Default Value: On
;   Development Value: Off
;   Production Value: Off

; request_order
;   Default Value: None
;   Development Value: "GP"
;   Production Value: "GP"

; session.gc_divisor
;   Default Value: 100
;   Development Value: 1000
;   Production Value: 1000

; session.hash_bits_per_character
;   Default Value: 4
;   Development Value: 5
;   Production Value: 5
```

```

; short_open_tag
;   Default Value: On
;   Development Value: Off
;   Production Value: Off

; track_errors
;   Default Value: Off
;   Development Value: On
;   Production Value: Off

; url_rewriter.tags
;   Default Value: "a=href,area=href,frame=src,form=,fieldset="
;   Development Value: "a=href,area=href,frame=src,input=src,form=fakeentry"
;   Production Value: "a=href,area=href,frame=src,input=src,form=fakeentry"

; variables_order
;   Default Value: "EGPCS"
;   Development Value: "GPCS"
;   Production Value: "GPCS"

```

If you are installing from a precompiled distribution you might see default standards, and the `php.ini` file might be split into multiple files for easier management.

In addition to the `display_errors` and `error_reporting` there is an option called `log_errors`. When it's turned on all the errors that are encountered based on the `error_reporting` flag will be written to a log file. This is very useful when trying to track down errors on a development or production server after they happened. As a developer you might not have access to the browser where the error occurred so being able to read through the errors after the fact can be a helpful debugging tool. Along with the `log_errors` option there is also an option (`error_log`) to specify the name of the error log file. If it's not specified the errors will go into Apache's error log.

The `error_log` option can be set to a file name or a full path. When it's configured as just a file name the file will be placed in the same directory as where the main PHP script of the request was found. If you use a full path the error file is written at a specific location and it can even be shared among multiple hosts on the same server.

If the `E_NOTICE` option is turned on the error log file can grow very fast. It is recommended to clear these files from time to time.

The option `register_argc_argv` is used to define whether the variables `$argc` and `$argv` are defined. These values are commonly used with the CLI version of PHP to access the number of arguments passed (`$argc`) and the values of these arguments (`$argv`). The CLI version will actually overwrite this value in the `php.ini` and always turn these values on.

The value makes little sense in a web environment where the parameters to the script are stored in `$_POST`, `$_GET`, and other super global variables.

The default `php.ini` files are about 68kb in size. They include quite a bit of descriptive text that is simply ignored when the file is parsed. Big or small versions of the `php.ini` file do not make any noticeable difference as the file is only read once when the server starts up.

In addition to the values described here it's also important to look at configuration values for memory usage, execution time, size of POST, and upload data. If the max memory limit is set too high it could be possible to run out of resources in a high traffic situation. On the other hand the memory limitation must be high enough to handle the data. It is important to code the site in way that makes it possible to stay within reasonable limits. If you are running into a memory problem you could just go ahead and increase the configured value, but that might be an indication that you need to refactor the script to be more efficient with the resources.

Memory allocation can only be configured in `php.ini` (or in the apache configuration). That is not the case for the `max_execution_time`. It is set to 30 seconds by default and that should be enough for most of the requests on any site. In fact any request that takes more than 4 seconds should be optimized if possible to give the user the best possible experience. It can however be necessary to increase the execution time on certain request. If you have a query that analyzes a large data set and needs additional time you can specify the time needed with the use of the `ini_set()` function. The syntax to extend the max execution time to 90 seconds look like this:

```
ini_set('max_execution_time', 90);
```

The web server might have its own timeout values that will interrupt the execution of a PHP script if this value is exceeded. The `max_execution_time` only affects the actual PHP runtime. If the PHP script contains system calls the 'clock' is stopped while these calls are running.

One important configure option is the default time zone. Without this configuration the system will generate a warning each time one of the date functions are used. The setting is called `date.timezone` and can be set to any of the supported time zone values. Setting it to UTC will cause all the date/time functions to work the same way as the once prefixed with `gm`. The function `date()` and `gmdate()` will return the same value. Setting it to any other value like 'America/Los_Angeles' will cause the two functions to return values with an eight hour offset.

The script when needed can change the default time zone. If a user's time zone is stored in a database the script can use that value to set the time zone. That way all representation of data/time values can be done according to each user's time zone. The function to set the time zone dynamically is called `date_default_timezone_set()` and it takes a string as the only parameter.

Most of the configuration options apply to both Windows and Linux, but a few of them are specific to Windows. The big difference is things like naming of extensions. On linux the names end in `.so` and on Windows they are prefixed with `php_` and end in `.dll` (`mysql.so` vs. `php_mysql.dll`).

Extensions can be loaded by the following configurations:

```
extension=mysql.so
```

or on Windows

```
extension=php_mysql.dll
```

Extensions can also be compiled in so there is no need to load them in `php.ini`. This is the case with the standard extension (Core, ctype, date, etc.).

Recipe 1-3. Compiling PHP

Problem

Using a precompiled version of PHP is most likely going to be good enough for most PHP developers, but if the operating system doesn't provide the latest version or you want to create your own extensions or perhaps even contribute to the PHP project, you will have to compile PHP from source code. How is this done?

Solution

There are two basic ways to get the source code for PHP. The first is to download a preconfigured and officially released version of PHP, and the other is to get the source code from the git repository. The first option provides all the files matching a specific version and when a new version is released the developer must download that package and compile everything again.

Using the git repository makes it very easy to switch between versions and also very easy to always have the latest version. It is not recommended to run a production server on the current development branch. It might contain untested features. Production servers should be run on the officially released versions, as these have been through religious QA testing.

In order to compile PHP from sources it's necessary to have a few tools installed. On a Linux environment these include automake, autoconf, libtool, and the compiler (gcc). On windows you will need a version of Microsoft Visual Studio. In addition to these there are a number of open source libraries that PHP depends on. Most of these can be detected during installation and quickly installed from the package manager (on Linux). On Windows there is no package manager that makes this easy, but it's still doable to download the source packages for these libraries and compile them as needed.

How it Works

The main difference between using an official release of PHP and compiling from the git repository is the use of the 'buildconf' utility. On the official releases this was done when the source files were packaged, and if you try to run the command it will generate a warning. The basic functionality of the buildconf script is to scan through all the directories in the source tree and generate the configure file. The configure file is then used to generate the Makefile, the input to the compiler that describes all the files and dependencies needed to compile the configured options. This section will demonstrate building PHP 7 from a cloned git repository.

On a clean install of CentOS 7 the first step will be to a version of git. Use the package manager to do this:

```
$ yum install git
```

Make sure you are running this command as root or remember to prefix it with sudo. Depending on the packages installed on the system already, this might install a number of other packages that git depends on.

In the home directory create a folder called Source and change to that folder and then use the git command to clone the git repository:

```
$ mkdir Source
$ cd Source
$ git clone http://git.php.net/repository/php-src
```

This will create a directory called php-src with a copy of all the source files. If you need a specific version of PHP use the `git checkout` command to switch to the specific branch. The command `git branch -a` will give a list of all available branches. The list is quite long as the repository contains versions going back to PHP 4.0.

By default the master branch is checked out. This branch contains the latest and greatest code. As of today that corresponds to the unreleased PHP 7.1 version. To get the latest released version (PHP 7.0) uses this command:

```
$ git checkout PHP-7.0
Branch PHP-7.0 set up to track remote branch PHP-7.0 from origin.
Switched to a new branch 'PHP-7.0'
```

Before the buildconf command is executed it's necessary to make sure the toolchain is complete.

```
$ yum install gcc autoconf automake libtool
```

This will install enough tools to run the buildconf command:

```
$ ./buildconf
buildconf: checking installation...
buildconf: autoconf version 2.69 (ok)
rebuilding aclocal.m4
rebuilding configure
rebuilding main/php_config.h.in
```

This will generate the configure script that is used to specify the operation to use when compiling PHP. Running this command at this time will result in a long output that ends with an error:

```
$ ./configure
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
...
configure: WARNING: This bison version is not supported for regeneration of the Zend/PHP
parsers (found: none, min: 204, excluded: ).
checking for re2c... no
configure: WARNING: You will need re2c 0.13.4 or later if you want to regenerate PHP parsers.
configure: error: bison is required to build PHP/Zend when building a GIT checkout!
```

This error indicates that the system is missing some functionality required to compile PHP. In this case it's the bison library. This can be installed with the following command:

```
$ yum install bison
```

After that we can run the configure script again and look for more errors. This will be the missing libxml2 library. By default PHP includes support for SimpleXML and other XML extensions. Installing the library alone is not quite enough. The system require both the library and the header and other files needed to link against so the command to install looks like this:

```
$ yum install libxml2-devel
```

In this example the configure script was used without any options. If options to include various extensions were used, other dependencies could show up and you will have to install these and repeat the configure option until all the dependencies are installed.

When the configure script completes successfully the output will end with these lines:

```
Generating files
configure: creating ./config.status
creating main/internal_functions.c
creating main/internal_functions_cli.c
+-----+
| License:                                     |
| This software is subject to the PHP License, available in this |
| distribution in the file LICENSE. By continuing this installation |
| process, you are bound by the terms of this license agreement. |
| If you do not agree with the terms of this license, you must abort |
| the installation process at this point.       |
+-----+
```

Thank you for using PHP.

```
config.status: creating php7.spec
config.status: creating main/build-defs.h
config.status: creating scripts/phpize
config.status: creating scripts/man1/phpize.1
config.status: creating scripts/php-config
config.status: creating scripts/man1/php-config.1
config.status: creating sapi/cli/php.1
config.status: creating sapi/cgi/php-cgi.1
config.status: creating ext/phar/phar.1
config.status: creating ext/phar/phar.phar.1
config.status: creating main/php_config.h
config.status: executing default commands
```

The configuration is now ready to build by running the make command. This takes a few minutes depending on the number of enabled extensions. The make command will produce many lines of output ending with these lines:

```
Build complete.
Don't forget to run 'make test'.
```

It is always a good idea to run the test suite before installing. It will provide valuable feedback to the QA team by sending a list of failed tests.

The new version of PHP is now ready to be installed. The default install location is in `/usr/local` because no other location was provided to the configure script. This will actually allow the installation of this newly compiled version alongside the version installed with the CentOS package manager. To install this version run this command:

```
$ sudo make install
```

After installation check the version of PHP:

```
$ php -v
PHP 7.0.6-dev (cli) (built: Apr  2 2016 20:05:26) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

So far it's only the CLI and CGI versions of PHP that were compiled. In order to compile the Apache module version it's necessary to add at least one option to the configure command:

```
$ ./configure --with-apxs2
```

This command will most likely fail, as the `apxs` script is not installed by default with the `apache` package from CentOS. The `apxs` script is installed with the `httpd-devel` package.

```
$ sudo yum install httpd-apxs2
$ ./configure --with-apxs2
$ make
$ sudo make install
```



This will install the php module into the httpd.conf file located in /etc/httpd. Because the PHP 5.4 version was installed earlier there is now a conflict between the two and it's no longer possible to start the Apache server without errors. To resolve these errors it's necessary to edit the file to disable the PHP 5 version of the module. The install script installs the new module in /etc/httpd/conf/httpd.conf and the CentOS installation created a modules configuration file /etc/httpd/conf.modules.d/10-php.conf. First edit the file /etc/httpd/conf/httpd.conf and locate the line

```
LoadModule php7_module          /usr/lib64/httpd/modules/libphp7.so
```


This line should be removed and added to the file /etc/httpd/conf.modules.d/10-php.conf. The file will now look like this:

```
#
# PHP is an HTML-embedded scripting language which attempts to make it
# easy for developers to write dynamically generated webpages.
#
<IfModule prefork.c>
# LoadModule php5_module modules/libphp5.so
  LoadModule php7_module          /usr/lib64/httpd/modules/libphp7.so
</IfModule>
```

The Apache server can now be started and pointing a browser to the phpinfo.php script will show the following information:

PHP Version 7.0.6-dev		
System	Linux localhost.localdomain 3.10.0-327.13.1.el7.x86_64 #1 SMP Thu Mar 31 16:04:38 UTC 2016 x86_64	
Build Date	Apr 2 2016 20:34:15	
Configure Command	'./configure' '--with-apxs2'	
Server API	Apache 2.0 Handler	
Virtual Directory Support	disabled	
Configuration File (php.ini) Path	/usr/local/lib	
Loaded Configuration File	(none)	
Scan this dir for additional .ini files	(none)	
Additional .ini files parsed	(none)	
PHP API	20151012	
PHP Extension	20151012	
Zend Extension	320151012	
Zend Extension Build	API320151012,NTS	
PHP Extension Build	API20151012,NTS	
Debug Build	no	
Thread Safety	disabled	
Zend Signal Handling	disabled	
Zend Memory Manager	enabled	
Zend Multibyte Support	disabled	
IPv6 Support	enabled	
DTrace Support	disabled	
Registered PHP Streams	php, file, glob, data, http, ftp, phar	
Registered Stream Socket Transports	tcp, udp, unix, udg	
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk	

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies



The list of modules installed can be shown by running the CLI version of php with the parameter `-m`.

```
$ php -m
[PHP Modules]
Core
ctype
date
dom
fileinfo
filter
hash
iconv
json
libxml
mysqli
mysqlnd
pcre
PDO
pdo_sqlite
Phar
posix
Reflection
session
SimpleXML
SPL
sqlite3
standard
tokenizer
xml
xmlreader
xmlwriter

[Zend Modules]
```

Recipe 1-4. Installing MySQL

Problem

We now have a system with a web server that supports executing PHP scripts and returning the output of the script instead of the raw source file. In order to complete the installation of a fully functional development stack we need to install the MySQL database server.

Solution

The basic installation of a MySQL compatible database on CentOS 7 is called MariaDB. The official MySQL database is owned by Oracle. It is still kept as an open source and they have a few community-based versions. When Oracle purchased Sun Microsystems and thereby acquired the MySQL source code the original development team created a new version. They have been developing new features and improved performance. So far the two products provide a feature set that is compatible but as time goes by we might see the two products develop features that are not compatible.

To install the MariaDb Server and all its dependencies, run these commands as the root user:

```
$ yum install mariadb-server
$ systemctl enable mariadb
$ systemctl start mariadb
```

After that it is recommended to run the command `mysql_secure_installation`. This is a script that will set a root password, disable remote root login, and perform other changes that will enhance security. With the exception of setting a root password it is safe to answer yes to all questions. The name of the script is the same for both MySQL and MariaDb installations.

How it Works

This completes the installation of the database sever but there is no support in PHP for this database. In order to get the MySQLi extension installed it is necessary to reconfigure PHP to add the extension, recompile, and reinstall:

```
$ ./configure --with-apxs2 --with-mysqli
$ make
$ sudo make install
```

In this environment it is necessary to re-edit the `/etc/httpd/conf/httpd.conf` file and remove the entry added for the `php7_module`.

The PHP installation is now complete. If other extensions are needed the steps above can be repeated.

To install MariaDB on Windows is equally easy. The latest installation packages can be found at <https://mariadb.org/download/>, or if you prefer the MySQL version you can download the installation files from <http://www.mysql.com/downloads/>.

Recipe 1-5. Virtual Machines

Problem

When development is done on a Windows or Mac OSX computer and deployment is done on a Linux environment it is easy to run into issues that are related to differences in the operating systems, or there could be issues with lack of support for specific extensions on one of the platforms. Is there a way to minimize these problems?

Solution

There are a number of cloud-based services that offer relatively cheap virtual Linux environments. Many of these can be configured and launched in a few minutes and the user only pays for the time the configuration is active. The cost is as low as \$10 per month.

Having a virtual server in the cloud requires an Internet connection to use with the server. If work is done offline it is more convenient to install a Virtual Machine on the development environment. A Virtual Machine is a system that allows the primary operating system on a computer to be the host for one or more client systems. This is supported on Windows, Mac OSX, and Linux systems, either with native software, third-party freeware (VirtualBox), or commercial products (VM-Ware or Parallels).

How it Works

VirtualBox is supported by Oracle and can be downloaded for free at <https://www.virtualbox.org/>. After installation of the software you can create as many virtual machines as you have space on the hard drive. Each virtual machine requires 10-100Gb of disk space. It is possible to install both Windows and Linux guests. The guests can be installed from an ISO image that can be downloaded from the preferred Linux distributions web site. Windows even provides distributions of prebuilt systems that can be downloaded and launched. Most of these can be used for up to 90 days for testing purposes. If you have a valid Windows license you can also create an installation that doesn't expire.