Creating a Raspberry Pi LAMP server

Introduction

This tutorial expects that you already have a Raspberry Pi setup and ready to install your Linux Apache MySQL PHP (LAMP) environment. The article **creating a Raspberry Pi web server** details how to:

- **install Raspbian Linux**
- **familiarise yourself with the command line and text editor**
- **performance tuned the OS**
- **setup networking**
- **enabled SSH and SFTP**
- **have made your OS headless**
- **have made your server secure**
- **have gone through a server clean up exercise**
- **changed your hostname**
- **can access you server via the Internet**
- **have setup a domain name**

Install Apache and PHP

The Apache webserver is available to download from the Debian repositories. This can be done through the apt tools.

First, have you refreshed the software repositories? If not run `sudo apt-get update` to make sure that it knows about any new packages/versions available.

The following commands will install Apache and PHP version 5:

```
sudo apt-get install apache2 php5 libapache2-mod-php5
```

Next, create an extra file in the `/etc/apache2/conf.d` directory as follows:

```
sudo nano /etc/apache2/conf.d/servername.conf
```

Type your `ServerName` into the file under the `ServerName` directive:

```
ServerName    tariqkhan.rpi
```

As `Apache2` loads data from the files in `conf.d` into its configuration, these changes will remove the error concerning `Apache2` not finding your server name. It is preferable to use this method of adding to the Apache2 configuration as editing the main configuration file may introduce unintentional errors.

Reload the server using: `sudo service apache2 restart`

If you want to take a look at your PHP configuration, create the following file:

```
sudo nano /var/www/phpinfo.php
```

Inside that file put the following code: `<?php phpinfo(); ?>`

Save it and point your browser to http://domain.or.ip/phpinfo.php and this should detail configuration and settings.

Configuration files used by Apache:

- Apache main configuration file: **/etc/apache2/apache2.conf**

- Virtual domains: **/etc/apache2/sites-enabled/domain**
- Additional configuration directives: **/etc/apache2/conf.d/**
- Ports to listen to: **/etc/apache2/ports.conf**

Set permissions on the web directory

It is useful to change the permissions on the web directory (`www`) to allow your user to update the webpages without needing to be the root user.

Change the directory owner and group `sudo chown www-data:www-data /var/www`

Allow the group to write to the directory `sudo chmod 775 /var/www`

Add the user to the www-data group `sudo usermod -a -G www-data tariq`

You should logout and then log back in to pick up group permissions, or you can just start a new terminal.

Managing Apache2 modules

As an example, we will be enabling `mod-rewrite`, but the same rules apply for any other module.

Compiled statically inside the Apache2 binary:

```
core, http_core, prefork/worker/perchild, mod_access, mod_auth,
mod_log_config, mod_logio, mod_env, mod_setenvif, mod_mime,
mod_status, mod_autoindex, mod_negotiation, mod_dir, mod_alias,
mod_so
```

You can get the list of compiled in modules from the command line: `apache2 -l`

Apache2 modules installed and ready to be enabled:

```
actions, asis, auth_anon, auth_dbm, auth_digest, auth_ldap, cache,
cern_meta, cgi, cgid, dav, dav_fs, deflate, disk_cache, expires,
ext_filter, file_cache, headers, imap, include, info, ldap,
mem_cache, mime_magic, proxy, proxy_connect, proxy_ftp, proxy_http,
rewrite, speling, ssl, suexec, unique_id, userdir, usertrack,
vhost_alias
```

All the loading and configuration related entries are found in individual files inside the folder `/etc/apache2/mods-available/`. Here, we will find files like `module_name.load` (and if needed `module_name.conf`). Also, all additional installed modules will place their configuration files in the same place.

Inside the folder `/etc/apache2/mods-enabled/` we will find all the enabled modules. Here we will find `symlinks` to the files from `mods_available` for all the enabled modules.

`a2enmod` enables an Apache2 module (this does nothing else but creates the proper links to the module `.load` and `.conf` files). For example to enable the rewrite module:

```
sudo a2enmod rewrite
```

`a2dismod` disables an Apache2 module (removes the links from mod-enabled for the module). For example to disable the rewrite module:

```
sudo a2dismod rewrite
```

Giving Apache access to the file system

It is prudent to limit Apache's view of the file system to only those directories necessary. Start by denying access to everything then grant access to the necessary directories:

```
sudo nano /etc/apache2/sites-available/default
```

Deny access completely to file system root (`/`) then grant permissions as the default:

```
<Directory />

Options None

AllowOverride None

</Directory>
```

We then need to edit the default Apache2 configuration to set the default location of system web pages, allow access and AllowOverrides from .htaccess files or any rewrite rules created by websites. Find the following:

```
DocumentRoot /var/www

<Directory /var/www/>

Options -Indexes FollowSymLinks MultiViews

AllowOverride all

Order allow,deny

allow from all

</Directory>
```

Create a Virtual Host

Navigate to the location where Apache2 will read configuration files containing our setup for a virtual host: `cd /etc/apache2/sites-available`

You can place all virtual host configuration files here. But each file will not be enabled unless a symbolic link to the file is created in the parallel directory `sites-enabled`. There is a tool for doing this, which we will use once we have created a virtual host configuration file: `sudo nano oursites.conf`

Note that the `oursites` is plural, as we may have more than one virtual site! Then edit the file so that it contains the following text:

```
# oursites.conf



# First, the listening port (if not specified elsewhere):

Listen 80



# Next, the IP address and port for the virtual host. This assumes

# that you have only one IP address and port for this server.

# Be sure to substitute your own parameters throughout this file!
```

```
NameVirtualHost 10.0.0.97:80


# Next, add the default server, because creating a virtual host
# causes Apache2 to ignore the default server configured in the
# /etc/apache2/sites-available/default file.
# If you do not do this, any html files in /var/www will be ignored!
<VirtualHost 10.0.0.97:80>
DocumentRoot /var/www
DirectoryIndex index.htm index.html index.php
</VirtualHost>


# Next your first virtual server details:
<VirtualHost 10.0.0.97:80>
ServerAdmin email@youraddress.com
ServerName oursite1
ServerAlias samesite1
DocumentRoot /var/www/oursite1
DirectoryIndex index.htm index.html index.php
ErrorLog /var/www/oursite1/log/error.log
CustomLog /var/www/oursite1/log/access.log
</VirtualHost>


# This will allow you to access your "oursite1" website by that
name.
```

There are one or two other configuration adjustments to make before this will work. First, create a directory for the Apache2 log files, as given in the oursites.conf file: `mkdir /var/www/oursite1/log`

Then we must make a link to our new virtual host file in sites-enabled: `sudo a2ensite oursites.conf`

This program creates the link for you. Its sister program `a2dissite` will remove the link if you wish to take your virtual host site offline.

Finally, it is necessary to enter your new virtual host site name into your Raspberry Pi `/etc/hosts` file, your PC workstation `C:\Windows\System32\drivers\etc\hosts` file, or into your domain name server (DNS) if you have one.

Finally, restart Apache daemon: `/etc/init.d/apache2 restart` or `service apache2 restart`.

Enabling SSL

To enable SSL:

```
sudo a2enmod ssl

sudo a2ensite default-ssl
```

If you want to use self-signed certificates, you should install the `ssl-cert` package detailed in 'Creating self-signed certificates' below. Otherwise, just adjust the SSLCertificateFile and SSLCertificateKeyFile directives in `/etc/apache2/sites-available/default-ssl` to point to your SSL certificate. Then restart apache: `sudo /etc/init.d/apache2 restart`.

The SSL key file should only be readable by `root`, the certificate file may be globally readable. These files are read by the Apache parent process which runs as `root`. Therefore, it is not necessary to make the files readable by the `www-data` user.

Creating self-signed certificates

If you install the ssl-cert package, a self-signed certificate will be automatically created using the hostname currently configured on your computer. You can recreate that certificate (e.g. after you have changed `/etc/hosts` or DNS to give the correct hostname) as user root with:

```
make-ssl-cert generate-default-snakeoil --force-overwrite
```

To create more certificates with different host names, you can use

```
make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /path/to/cert-file.crt
```

This will ask you for the hostname and place both SSL key and certificate in the file `/path/to/cert-file.crt`. Use this file with the SSLCertificateFile directive in the Apache config (you don't need the SSLCertificateKeyFile in this case as it also contains the key). The file `/path/to/cert-file.crt` should only be readable by `root`. A good directory to use for the additional certificates or keys is `/etc/ssl/private`.

SSL workaround for MSIE

The SSL workaround for MS Internet Explorer needs to be added to your SSL VirtualHost section (it was previously in `ssl.conf` but caused `keepalive` to be disabled even for non-SSL connections):

```
BrowserMatch "MSIE [2-6]" \

nokeepalive ssl-unclean-shutdown \

downgrade-1.0 force-response-1.0

BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown
```

The default SSL virtual host in `/etc/apache2/sites-available/default-ssl` already contains this workaround.

Install the APC support package for PHP

APC is an alternative caching support system for PHP, which enables PHP intermediate code to be cached. This can improve the performance of Apache2 and other programs which may execute PHP code. Execute the following command to obtain and install the APC support system for PHP:

```
sudo apt-get install php-apc

sudo apt-get install php-pear php5-dev apache2-prefork-dev build-
essential make && pecl install apc
```

Then edit the `php.ini` configuration file: `sudo nano /etc/php5/apache2/php.ini` and add the following text to the file, in the Dynamic Extension section, some way down the file: `extension=apc.so`

Then to take account of system and configuration changes, stop and start the Apache2 web server:

```
service apache2 start

service apache2 stop
```

...or `service apache2 restart`.

Install MySQL

MySQL is the most popular database server, whilst there are other alternatives some of which may require less resources, most third party software for Linux is designed to use MySQL. Remember running MySQL server, to a fair extent, requires at least 256mb of RAM in your server. If you don't need a database then you can skip this.

The MySQL server and MySQL client is also available through the Debian repositories and installed as:

```
sudo apt-get install mysql-server mysql-client php5-mysql
```

During the install there is a prompt request for a password. The password is for the MySQL root user.

The configuration file of MySQL is located at: `/etc/mysql/my.cnf`

Creating MySQL users and Changing Root Password

By default MySQL creates a user as root and runs with no password. If you want to change the root password:

```
mysql -u root

mysql> USE mysql;

mysql> UPDATE user SET Password=PASSWORD('new-password') WHERE
user='root';
```

```
mysql> FLUSH PRIVILEGES;
```

You must **never use root password**, so you will want to create a user so that you can connect to MySQL database for a PHP script. Alternatively, you can add users to MySQL database by using a control panel like phpMyAdmin to easily create or assign database permission to users.

Install phpMyAdmin

phpMyAdmin is a nice web based database management and administration software and easy to install and configure under Apache. All you need to do is enter: `sudo apt-get install phpmyadmin`

The package will begin installing. You will be asked which web server is installed, choose `apache2`.

Next we'll need to configure phpMyAdmin's database. When prompted, choose `Yes`. Next you'll be asked for an administrative password, this is the root password that was set during the MySQL installation. You'll be asked to set a password for MySQL. I've used the same password as the MySQL root password, but it is up to you what you set here.

The phpMyAdmin configuration file is located at: `/etc/phpmyadmin`

Configure Apache to work with phpMyAdmin

Next we need to change the Apache configuration to allow us to use http://your.raspberrypi.domain/phpmyadmin to access it. To do this, enter the following command to alter the configuration:

```
sudo nano /etc/apache2/apache2.conf
```

Navigate to the bottom of the file (keep pressing `CTRL` and `V` simultanously to jump pages until you're at the bottom of the file) and add the following new line to the file: `Include /etc/phpmyadmin/apache.conf`

Now restart Apache: `/etc/init.d/apache2 restart`

Point your browser to: http://your.raspberrypi.domain/phpmyadmin

That's it! MySQL and phpMyAdmin are ready. Log in with your MySQL root password and create users to connect to database from your php script.

Documentation

The full Apache 2 documentation can be found on the web at **http://httpd.apache.org/docs/2.2/**

Some hints about securing Apache 2 on Debian are available at
http://wiki.debian.org/Apache/Hardening

Examples of Virtual host configurations available at
http://httpd.apache.org/docs/current/vhosts/examples.html