# Building Web Applications with LAMP

Jason W. Dixon

2003-02-12

**Abstract**

Today's Internet is an evolving medium. In order to maintain the necessary traffic to sustain one's presence, most commercial websites, and, indeed, many private ones as well, rely on "fresh" information to stimulate and capture their readers' interests. The ability to maintain the necessary balance between dynamic content and a scalable, secure site has forced most developers to migrate away from the static pages of the past to modern, scalable designs. The obvious solution was to use HTML (HyperText Markup Language) as it was intended; HTML is a structural markup language that is used to convey most of the information found on the World Wide Web. However, HTML, coupled with modern scripting languages, allows us to provide customized, "on-the-fly" pages for our clients.

The ability of this framework to provide highly customized solutions have enabled developers to take further advantage of the client/server model as the basis for traditional "thick" applications. Coupled with powerful RDBMS servers, such as the MySQL database, these applications can be extended to solve a spectrum of solutions, efficiently and inexpensively.

Naturally, open standards such as those found on the Internet lead us towards open solutions. Linux, the free operating system developed by Linus Torvalds and maintained by thousands of volunteers worldwide, provides an open platform on which to build our dynamic systems. All of today's leading Internet protocols and server software has been- or will soon be- ported to Linux. Apache, the world's leading webserver software, has been a mainstay on Linux servers for many years. Apache allows us to quickly and securely serve up static or dynamic content.

This talk presents a fully open-sourced solution for creating your own Web Application using LAMP. While the concept of LAMP has been generalized to include the various *BSD operating systems, PostgreSQL database, and other free projects, LAMP is rooted in the technologies of Linux, Apache, MySQL, and Perl/PHP/Python. We will be building our application using Red Hat Linux 8.0, Apache 2.0.40, MySQL 3-23.54, and Perl 5.8.0.

# 1 Linux

## 1.1 Introduction

Red Hat Linux is the leading commercial Linux distribution. Although their visible product is the release of the GNU/Linux operating system, their primary focus is on services and support. This presentation is based on a custom installation of Red Hat Linux 8.0. It is assumed that the audience is familiar with the installation process of the base Red Hat Linux system.

## 1.2 Prerequisites

The following packages will need to be installed. Each can be queried with the "rpm -q <package>" command. If they are not installed, install them from CD[1] with the "rpm -ivh </path/to/rpm>" command.

1. Apache
2. Perl
3. mod_ssl
4. mod_perl[2]
5. MySQL and MySQL-server[3]
6. perl-DBI
7. libdbi-dbd-mysql[4]

## 1.3 Security

There are a plethora of security issues to take into consideration when building a public webserver. First and foremost, the system should be patched or upgraded to fix any known exploits. Running "up2date" on Red Hat Linux will allow us to update all security and bug fixes to the most current versions.

- /etc/sysconfig/iptables

```
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 80 --syn -j ACCEPT
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 443 --syn -j ACCEPT
```
[5]

- /sbin/service iptables restart

[1] ... or a download mirror such as ftp://mirror.cs.wisc.edu/pub/mirrors/linux/redhat/8.0/en/os/i386/RedHat/RPMS

[2] Unfortunately, mod_perl 2.0 does not play nicely with Apache 2.0 as of the release of Red Hat Linux 8.0. It appears that the included Perl does not handle multiple invocations well.

[3] MySQL can be installed either from the RPM's included with the Red Hat Linux release, or from downloaded RPM's found at the MySQL AB website (http://www.mysql.com).

[4] This package might have a different name, depending on the source or distribution. Basically, this is the DBD (database driver) for MySQL.

[5] Only add access to TCP port 443 if you want to allow HTTPS traffic.

# 2 Apache

## 2.1 Configuration

While we are working with Apache 2.0, much of the configuration changes we're implementing are similar with the older Apache 1.3 series.

- /etc/httpd/conf/httpd.conf
- ServerName[6]

```
<Directory "/var/www/html/lamp">
Options ExecCGI
AllowOverride None
DirectoryIndex index.cgi
Order Allow,Deny
Allow from all
SSLRequireSSL[7]
AuthType Basic
AuthName LAMP
AuthUserFile /etc/httpd/conf/lamp.users
Require valid-user
</Directory>
```

- htpasswd -c /etc/httpd/conf/lamp.users guest[8]
- chown root.apache /etc/httpd/conf/lamp.users
- chmod 640 /etc/httpd/conf/lamp.users
- /sbin/chkconfig –level 345 httpd on
- /sbin/service httpd start

## 2.2 Testing

- netstat -vant
- telnet localhost 80
- GET /

[6]Set this directive with the name of your host ONLY if you have working DNS. This directive will allow Apache to perform a redirect to itself for URL completion purposes. If you do not have working forward DNS for your hostname, you can still use "localhost" for same-system testing, but comment this directive out when the system goes into production status.

[7]Include this directive to require HTTPS connections on port 443. Otherwise, leave it out.

[8]Only use the -c flag for creating a new password file.

# 3 MySQL

Although MySQL version 4.x is available, we will stay with the stable 3.x series. Many hard-core DBA's prefer PostgreSQL for it's advanced features (hot backups, transactions, etc.), but MySQL will easily meet the needs of most developers for non-commerce websites.

## 3.1 Configuration

- /sbin/chkconfig –level 345 mysqld on
- /sbin/service mysqld start
- mysql_install_db[9]
- mysqladmin -u root password <newpassword>
- mysqladmin -u root -p create <dbname>

## 3.2 Building the Data

- mysql -u root -p mysql

```
> CREATE TABLE data (
    name varchar(20) NOT NULL default '',
    phone smallint(4) NOT NULL default '0',
    PRIMARY KEY  (name),
    UNIQUE KEY name (name)
  ) TYPE=MyISAM;
> INSERT INTO data VALUES ('Dave',1001);
> INSERT INTO data VALUES ('Jason',1002);
> INSERT INTO data VALUES ('Perry',1003);
> INSERT INTO data VALUES ('Joe',1004);
> INSERT INTO data VALUES ('Linus',1337);
> GRANT SELECT on lamp.* to lampuser IDENTIFIED BY '<password>';
> GRANT SELECT on lamp.* to lampuser@localhost IDENTIFIED BY
>   '<password>';
> exit;
```

- mysqladmin -u root -p reload

## 3.3 Testing

- mysql -u lampuser -p lamp

```
> show tables;
> describe data;
> select * from data;
```

[9]Red Hat's installation script takes care of this task for us.

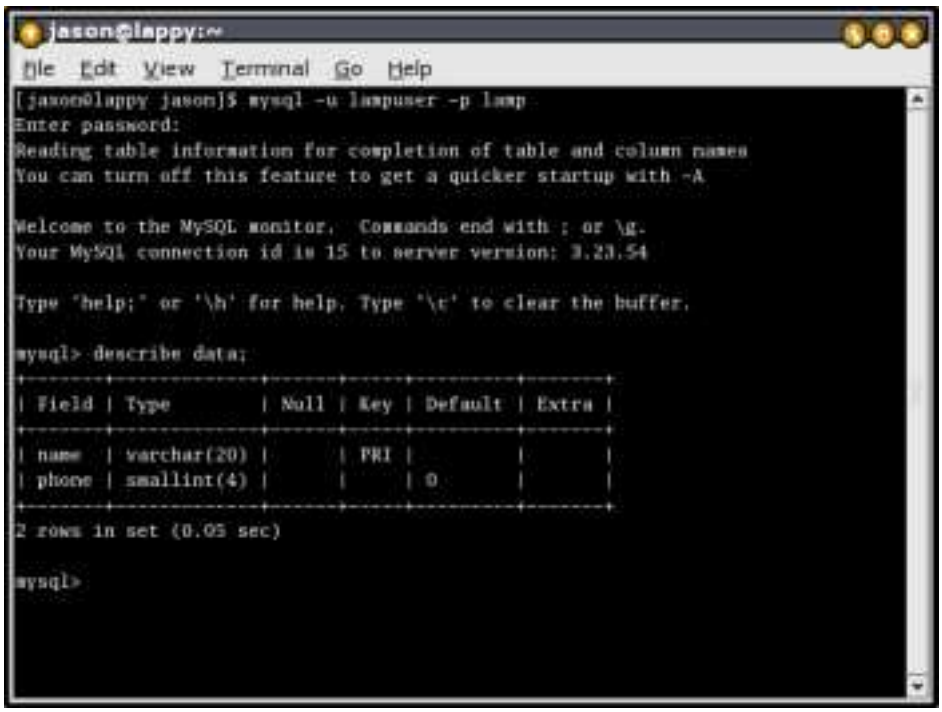
## 3.4 Backups

An easy way to perform nightly backups is with the mysqldump command via cron.

- mysqldump -u root -p lamp > lamp-‘date +%Y%m%d‘.sql
- mysql -u root -p lamp < lamp-20030212.sql

## 3.5 Client Software

### 3.5.1 mysql

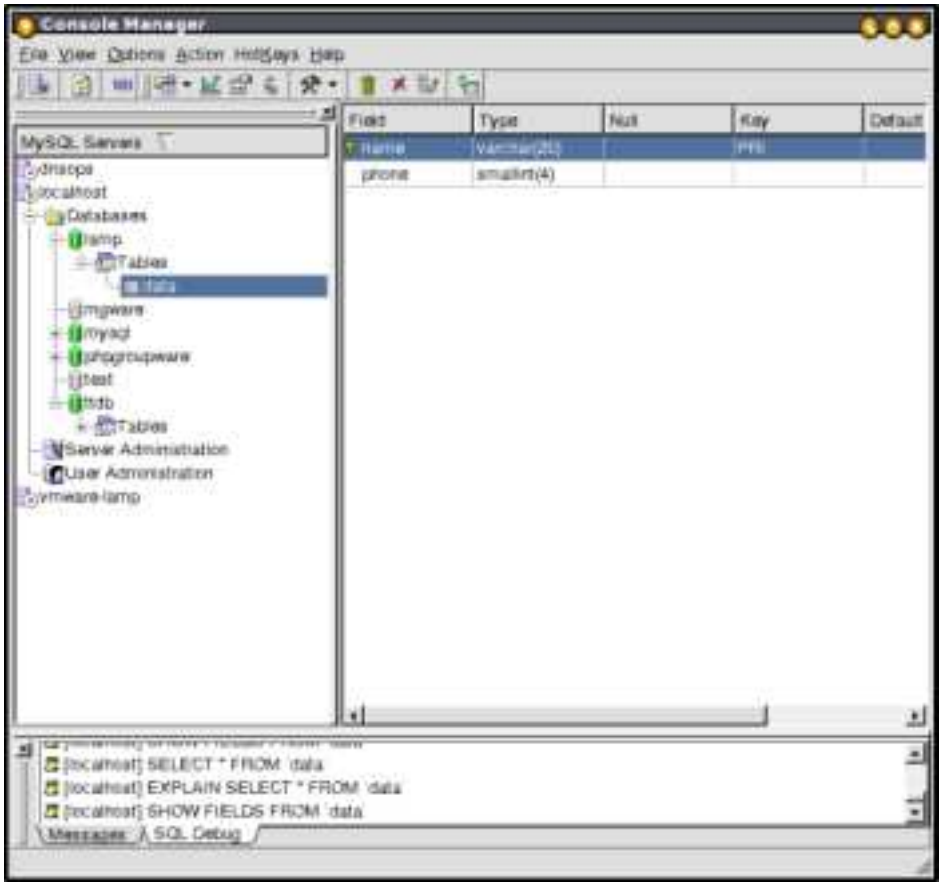



The command line client provided with the MySQL distribution. Provides many CLI creature comforts such as autocompletion, buffer history, and buffer search.

### 3.5.2 phpMyAdmin

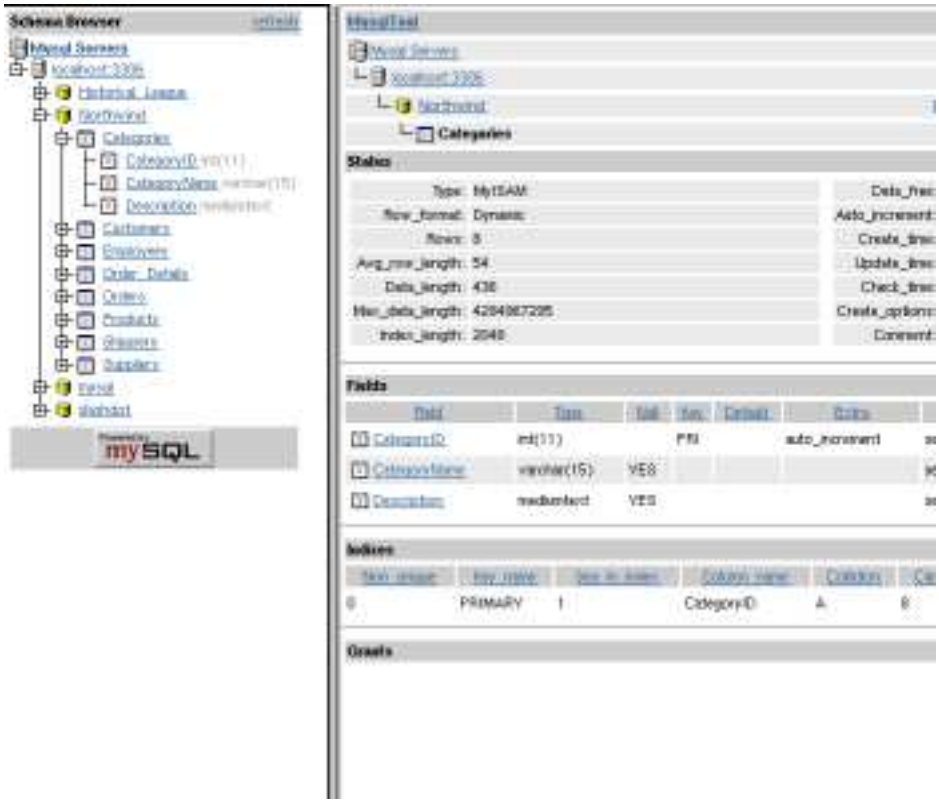
Manage databases and tables from your web browser. It requires PHP and a webserver.

### 3.5.3 MySQL Control Center




This is MySQL AB's own graphical front-end for MySQL servers. It is full-featured, free, and available in source or binary download for Linux or Windows clients.

### 3.5.4 MysqlTool



Yet another web-based administration tool. This one relies on the Perl CGI and DBI modules. It optionally supports encrypted connections via the Crypt::Blowfish module.

# 4 Perl

## 4.1 Test CGI

We would first like to verify that a simple perl CGI will work from our application directory.

- mkdir /var/www/html/lamp

Test script (test.cgi):

```
#!/usr/bin/perl
use strict;
use CGI;
my $cgi = new CGI;
```

```
print $cgi->header;
print $cgi->start_html;
foreach (keys %ENV) {
    print "$_ $ENV{$_}<br>";
}
print $cgi->end_html;
```

- https://localhost/lamp/test.cgi

## 4.2 Installing Perl Modules

- http://search.cpan.org/CPAN/authors/id/S/SA/SAMTREGAR/HTML-Template-2.6.tar.gz
- tar zxvf HTML-Template-2.6.tar.gz
- cd HTML-Template-2.6 && perl Makefile.pl && make && sudo make install

## 4.3 HTML::Template

The HTML::Template module is recommended for projects where the logic (Perl) and content (HTML) need to be segregated. Examples of this would include companies where the Web designers are not fluent back-end coders. A normal CGI script is created, but it does not directly create the HTML output. Instead, an HTML::Template object is created through the OO interface, and parameters are passed onto the requested template file (*.tmpl, by convention) via the param() object method. The only uncommon formatting you'll notice in the templates are the TMPL markup tags used to receive the passed values.

Example Tags:

```
<TMPL_VAR></TMPL_VAR>
<TMPL_LOOP></TMPL_LOOP>
<TMPL_IF>
<TMPL_ELSE>
</TMPL_IF>
<TMPL_UNLESS></TMPL_UNLESS>
<TMPL_INCLUDE></TMPL_INCLUDE>
```

While TMPL_VAR allows for variable substitution, the remaining majority of tags provide a control structure. TMPL_LOOP, for example, allows for easy creation of HTML tables. Simply pass in a reference to an array of hashes, and HTML::Template handles the rest, creating the table rows (or columns) as you define via HTML markup.

Sample Template:

```
<html>
<head>
```

```
<title><TMPL_VAR name="title"></title>
</head>
<body>
<h1><TMPL_VAR name="title"></h1>
<table>
<TMPL_IF contacts_loop>
    <TMPL_LOOP name="contacts_loop">
        <tr>
            <td><TMPL_VAR name="name"></td>
            <td><TMPL_VAR name="phone"></td>
        </tr>
    </TMPL_LOOP>
</TMPL_IF>
</table>
</body>
</html>
```

And the sample CGI (index.cgi) which calls the template:

```
1:  #!/usr/bin/perl
2:
3:  use strict;
4:  use DBI;
5:  use HTML::Template;
6:
7:  my $title = "Phone Contacts";
8:  my @contacts_loop;
9:  my $contacts = get_contacts();
10:
11: for my $key (keys %$contacts) {
12:         my %temp_hash = (
13:                 name => $key,
14:                 phone => $contacts->{$key},
15:         );
16:         push(@contacts_loop, \%temp_hash);
17: }
18: my $template = HTML::Template->new(
19:         filename => 'contacts.tmpl',
20:         die_on_bad_params => 0,
21: );
22: $template->param(
23:         title => $title,
24:         contacts_loop => \@contacts_loop
25: );
26: print "Content-Type: text/html\n\n";
27: print $template->output;
```

```
28:
29: sub get_contacts {
30:         my $dbh = DBI->connect("DBI:mysql:lamp:localhost","lampuser","<pass>")
31:                 || die $DBI::errstr;
32:         my $select_query = "select * from data";
33:         my $sth = $dbh->prepare($select_query) || die $dbh->errstr;
34:         $sth->execute() || die $dbh->errstr;
35:         my %data;
36:         while (my $result = $sth->fetchrow_hashref) {
37:                 $data{$result->{name}} = $result->{phone};
38:         };
39:         return \%data;
40: }
```

- Line 1 calls the Perl interpreter.
- Lines 3-5 require the strict pragma, and the DBI and HTML::Template modules.
- Lines 7-9 initialize 3 global variables. $contacts is a reference created by the return value from the subroutine get_contacts().
- Lines 11-17 create an array @contacts_loop which contains a hash reference for each key/value pair returned from the database query.
- Line 11 shows a dereference of $contacts to allow us to grab the keys from the hash.
- Lines 12-15 populate a temporary hash with the key/value pair.
- Line 16 adds the new hash to the @contacts_loop array.
- Lines 18-21 create a new Template object $template, using the "contacts.tmpl" file, and turning off the die_on_bad_params[10]debugging option.
- Lines 22-25 call the param() method on the $template object to pass the $title variable and a reference to an array of hashes @contacts_loop.
- Line 26 prints the Content-Type header.
- Line 27 sends the aggregated template to STDOUT.
- Lines 29-40 define the subroutine get_contacts(), which is used to gather the data from the database and return a hash reference.
- Lines 30-31 create a database handle object (connection) to the "lamp" database on localhost, using the configured username and password. Failure to connect will force a die() with the database error ($DBI::errstr).

[10]If set to 0 the module will let you call $template->param(param_name => 'value') even if 'param_name' doesn't exist in the template body. Defaults to 1.

- Line 32 creates a string $select_query containing our SQL query.
- Line 33 creates a statement handle object. This is used to prepare our SQL query $select_query.
- Line 34 executes our prepared query.
- Line 35 initializes a new hash. This hash will be used to contain all contact names and phone numbers retrieved from the database query.
- Lines 36-38 perform a control loop which gathers hash references to our data "while" data is still available from the statement handle.
- Line 37 assigns a new key/value pair in the %data hash based on the collected name/phone pair.
- Line 39 returns reference to the %data hash.

## 4.4 Testing

- Connect to https://localhost/lamp/ or https://localhost/lamp/index.cgi to verify the script works.

## 4.5 Advanced Topics

### 4.5.1 Security

Although this presentation has not focused extensively on Internet security, any developer planning or designing a web application should spend sufficient time researching and resolving predictable points of weakness in all seven OSI layers.[11]

### 4.5.2 Sessions

The ability to keep state between invocations of the same CGI or pages in an application is necessary for online commerce, communities, webmail, and much more. Methods for handling this include hidden form parameters and cookies. Most modern programming languages (Perl, PHP, Python) have modules available which assist the developer in maintain session state without dealing with the granular details.

### 4.5.3 CPAN

We've touched briefly on the use of Perl modules to facilitate new applications. The Comprehensive Perl Archive Network provides Perl developers with a resource to borrow and donate open-sourced Perl code.

[11] http://www.randywanker.com/OSI/

# 5 Conclusion

LAMP configurations serve as excellent cost-conscious platforms for today's web applications. Corporate engineers and casual hackers alike can take advantage of free software to build scalable, dynamic projects. Thanks to the open-source licensing and development of the LAMP components, security and bug fixes are often released much sooner than you would find in their proprietary counterparts. Combined, the facets of these projects have taken the Internet by storm, and are unlikely to be unseated in the forseeable future.

# References


[1] Red Hat Linux, http://www.redhat.com

[2] Apache, http://httpd.apache.org

[3] MySQL, http://www.mysql.com

[4] Perl, http://www.perl.org

[5] mod_SSL, http://www.modssl.org

[6] mod_perl, http://perl.apache.org

[7] HTML::Template, http://html-template.sourceforge.net

[8] PHP, http://www.php.net

[9] Python, http://www.python.org

[10] netfilter/iptables, http://www.netfilter.org

[11] PerlMonks, http://www.perlmonks.org

[12] CPAN, http://www.cpan.org

[13] phpMyAdmin, http://www.phpmyadmin.net

[14] MySQL Control Center, http://www.mysql.com/downloads/mysqlcc.html

[15] MysqlTool, http://www.dajoba.com/projects/mysqltool/