DEV SHED DOT COM

Introduction to mod_perl (part II): mod_perl Quickstart

By Stas Bekman

If you like this article, please make a donation to the Perl development grant fund.

Table of Contents

Introduction

In the previous article, I've shown quite amazing web performance reports from companies that have deployed mod_perl heavily. You might be surprised but you can quite easily get similarly amazing results if you move your service to mod_perl as well. In fact, getting started with mod_perl shouldn't take you more than 30 minutes, the time it takes to compile and configure the server on a decent machine and get it running.

In this article I'll show step by step installation and configuration scenarios, and chances are you will be able to run the basic statically compiled mod_perl setup without reading any other documents. Of course you will want and need to read the documentation later, but I think you will agree with me that it's ultimately cool to be able to get your feet wet without knowing much about the new technology up−front.

The mod_perl installation was tested on many mainstream Unix platforms, so unless you have some very non−standard system you shouldn't have any problems when building the basic mod_perl server.

If you are a Windows user, the easiest way is to use the binary package available from http://perl.apache.org/distributions.html. From the same location you can download the Linux RPM version and CVS snapshots. However I recommend to always build the mod_perl from source, and as you will see in a moment, it's an easy thing to do.

Installing mod_perl

So let's start with the installation process. If you are an experienced Unix user, you need no explanation for the following commands. Just copy and paste them and you will get the server installed.

I'll use a % sign as the shell program's prompt.

```
% cd /usr/src
% lwp−download
http://www.apache.org/dist/httpd/apache_1.3.20.tar.gz
% lwp−download
http://perl.apache.org/dist/mod_perl−1.26.tar.gz
% tar −zvxf apache_1.3.20.tar.gz
% tar −zvxf mod_perl−1.26.tar.gz
% cd mod_perl−1.26
% perl Makefile.PL APACHE_SRC=../apache_1.3.20/src \
DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
% make && make test && make install
% cd ../apache_1.3.20
% make install
```

That's all!

What's left is to add a few configuration lines to httpd.conf, an Apache configuration file, start the server and enjoy mod_perl.

If you have stumbled upon a problem at any of the above steps, don't despair −− the next section will explain in detail each and every step.

Installing mod_perl Detailed

If you didn't have the courage to try the steps in the previous section or you simply want to understand more before you try, let's go through the fine details of the installation process. If you have successfully installed mod_perl following the short scenario in the previous section, you can skip this section and move on to the next one.

Before we proceed, I should note that you have to become a root user in order to install the files in a protected area. If you don't have root access, you can install all the files under your home directory as well. We will talk about the nuances of this approach in a future articles. I'll also assume that you have perl and gcc or an equivalent C compiler installed.

I assume that all builds are being done in the /home/stas/src directory. So we go into this directory.

```
% cd /home/stas/src
```

Now we download the latest source distributions of Apache and mod_perl. If you have the LWP module installed (also known as libwww and available from CPAN), you should have the lwp−download utility that partly imitates your favorite browser by allowing you to download files from the Internet. You can use any other method to retrieve these files. Just make sure that you save both files in the /home/stas/src directory, as this will make it easier for you to follow the example installation process. Of course you can install both packages anywhere on your file system.

```
% lwp−download
http://www.apache.org/dist/httpd/apache_1.3.20.tar.gz
% lwp−download
http://perl.apache.org/dist/mod_perl−1.26.tar.gz
```

You can make sure that you're downloading the latest stable versions by visiting the following distribution directories: http://www.apache.org/dist/httpd/ and http://perl.apache.org/dist/. As you have guessed already, the former URL is the main Apache distribution directory, the latter is the same thing for mod_perl.

Untar both sources. You have to uncompress and untar the files. In addition to its main usage for tarring and untarring files, the GNU tar utility is able to uncompress files compressed by the gzip utility, when the −z option is used.

```
% tar −zvxf apache_1.3.20.tar.gz
% tar −zvxf mod_perl−1.26.tar.gz
```

If you have a non−GNU tar utility, chances are that it will be unable to decompress, so you need to do it in two steps. First uncompress the packages with:

```
% gzip −d apache_1.3.20.tar.gz
% gzip −d mod_perl−1.26.tar.gz
```

Then untar them with:

```
% tar −xvf apache_1.3.20.tar
% tar −xvf mod_perl−1.26.tar
```

If you don't have tar or gzip utilities available, either install them or use their equivalents.

Now go into the mod_perl source distribution directory.

```
% cd mod_perl−1.26
```

The next step is to create the Makefile.

```
% perl Makefile.PL APACHE_SRC=../apache_1.3.20/src \
DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
```

mod_perl accepts a variety of parameters, in this scenario we are going to use those that will allow you to do almost everything with mod_perl. Once you learn more about mod_perl you will be able to fine tune the list of parameters passed to Makefile.PL. In future articles I'll go through all the available options.

perl Makefile.PL ... execution will check for prerequisites and tell you which required software packages are missing from your system. If you don't have some of the Perl packages installed, you will have to install these before you proceed. They all are available from CPAN and can be easily downloaded and installed.

If you choose to install mod_perl with help of the CPAN.pm module, it will install all the missing modules for you. To do so, tell CPAN.pm to install the Bundle::Apache bundle.

This step also executes the ./configure script from Apache's source distribution directory (absolutely transparently for you), which prepares the Apache build configuration files. If you need to pass parameters to Apache's ./configure script, just pass them as options to perl Makefile.PL .... In future articles we will talk about all the available options.

Now you should build the httpd executable by using the make utility.

```
% make
```

This command prepares mod_perl extension files, installs them in the Apache source tree and builds the httpd executable (the web server itself) by compiling all the required files. Upon completion of the make process you get returned to the mod_perl source distribution directory.

make test executes various mod_perl tests on the freshly built httpd executable.

```
% make test
```

This command starts the server on a non−standard port (8529) and tests whether all parts of the built server function correctly. If something goes wrong, the process will report it to you.

make install completes the installation process of mod_perl by installing all the Perl files required for mod_perl to run and of course the server documentation (man pages).

```
% make install
```

You can use the following commands concatenation style:

```
% make && make test && make install
```

It simplifies the installation, since you don't have to wait for each command to complete before starting the next one. When installing mod_perl for the first time, it's better to do it step by step.

If you choose the all−in−one approach, you should know that if make fails, neither make test nor make install will be executed. If make test fails, make install will be not executed.

Finally, change to the Apache source distribution directory and run make install to create the Apache directory tree and install Apache header files (*.h), default configuration files (*.conf), the httpd executable and a few other programs.

```
% cd ../apache_1.3.20
% make install
```

Note that, as with a plain Apache installation, any configuration files left from a previous installation won't be overwritten by this process. You don't need to backup your previously working configuration files before the installation.

When the make install process completes, it will tell you how to start a freshly built web server (the path to the apachectl utility that is being used to control the server) and where the installed configuration files are. Remember or even better write down both of them, since you will need this information very soon. On my machine the two important paths are:

```
/usr/local/apache/bin/apachectl /usr/local/apache/conf/httpd.conf
```

So far we have completed the building and installation of the mod_perl enabled Apache. The next steps are to configure httpd.conf, write a little test script, start the server and check that the test script is working.

Configuring and Starting mod_perl Server

First thing first we want to make sure our Apache was built correctly and that we can serve plain HTML files with it. Why do that? To minimize the number of possible trouble makers, if we find out that mod_perl doesn't work. After you know that Apache can serve HTML files, you don't have to worry about it anymore. And if something goes wrong with mod_perl, you have eliminated the possibility that the httpd binary or basic configurations are broken, you know you are allowed to bind to the port you have configured your server to listen to, and that the browser you're testing with is just fine. Again, you should follow these guidelines when installing mod_perl for the first time.

Configure Apache as you always do. Set Port, User, Group, ErrorLog and other directives in the httpd.conf file (remember I've asked you to remember the location of this file at the end of the previous section?). Use the defaults as suggested, customize only when you have to. Values that you need to customize are ServerName, Port, User, Group, ServerAdmin, DocumentRoot and a few others. You will find helpful hints preceding each directive. Follow them if in doubt.

When you have edited the configuration file, it's time to start the server. One of the ways to start and stop the server is to use the apachectl utility. You start the server with:

```
% /usr/local/apache/bin/apachectl start
```

And stop it with:

```
% /usr/local/apache/bin/apachectl stop
```

Note that you have to be root when starting the server if the server is going to listen on port 80 or another privileged port (1024).

After you start the server, check in the error_log file (/usr/local/apache/logs/error_log is the file's default location) that the server has indeed started. Don't rely on the status apachectl reports. You should see something like the following:

```
[Thu Jun 22 17:14:07 2000] [notice] Apache/1.3.20 (Unix)
mod_perl/1.26 configured -- resuming normal operations
```

Now point your browser to http://localhost/ or http://your.server.name/ as configured with the ServerName directive. If you have set a Port directive with a value different from 80, apply this port number at the end of the server name. If you have used port 8080, test the server with http://localhost:8080/ or http://your.server.name:8080/ . You should see the infamous "It worked" page, which is an index.html file that make install in the Apache source tree installs for you. If you don't see this page, something went wrong and you should check the contents of the error_log file. You will find the path of the error log file by looking it up in the ErrorLog directive in httpd.conf.

If everything works as expected, shut the server down, open httpd.conf in your favorite editor, and scroll to the end of the file where we will add the mod_perl configuration directives (of course you can place them anywhere in the file).

Assuming that you put all scripts that should be executed by the mod_perl enabled server in the /home/httpd/perl/ directory, add the following configuration directives:

```
Alias /perl/ /home/httpd/perl/

PerlModule Apache::Registry
<Location /perl>
SetHandler perl-script
PerlHandler Apache::Registry
Options ExecCGI
PerlSendHeader On
allow from all
</Location>
```

Save the modified file.

This configuration causes every URI starting with /perl to be handled by the Apache mod_perl module. It will use the handler from the Perl module Apache::Registry.

Preparing the Scripts Directory

Now create a /home/httpd/perl/ directory if it doesn't yet exist. In order for you and Apache to be able to read, write and execute files we have to set correct permissions. You could get away by simply doing:

```
% chmod 0777 /home/httpd/perl
```

This is very very insecure and you should not follow this approach on the production machine. This is good enough when you just want to try things out and want to have as few obstacles as possible. Once you understand how things work, you should tighten the permissions of files served by Apache. In future articles we will talk about setting proper file permissions.

The "mod_perl rules" Apache::Registry Script

As you probably know, mod_perl allows you to re−use CGI scripts written in Perl which were previously used under mod_cgi. Therefore our first test script can be as simple as:

```
print "Content−type: text/plain\r\n\r\n";
print "mod_perl rules!\n";
```

Save this script in the /home/httpd/perl/mod_perl_rules1.pl file. Notice that the shebang line is not needed with mod_perl, but you can keep it if you want. So the following script can be used as well:

```
#!/usr/bin/perl
print "Content−type: text/plain\r\n\r\n";
print "mod_perl rules!\n";
```

Of course you can write the same script using the Apache Perl API:

```
my $r = shift;
$r−>send_http_header('text/plain');
$r−>print("mod_perl rules!\n");
```

Save this script in the /home/httpd/perl/mod_perl_rules2.pl file.

Now make both of the scripts executable and readable by the server. Remember that when you execute scripts from a shell, they are being executed by the user−name you are logged with. When instead you try to run the scripts by issuing requests, Apache needs to be able to read and execute them. So we make the script readable and executable by everybody:

```
% chmod 0755 /home/httpd/perl/mod_perl_rules1.pl \
/home/httpd/perl/mod_perl_rules2.pl
```

If you don't want other users to be able to read your script, you should add yourself into the groupname the webserver is running with (as defined by the Group directive) and then make the script owned by that group and then you can tighten the permissions. For example on my machine I run the server under the group httpd and I'm the only one who is in the same group, so I can do the following:

```
% chown stas.httpd /home/httpd/perl/mod_perl_rules1.pl \
/home/httpd/perl/mod_perl_rules2.pl
```

```
% chmod 0750 /home/httpd/perl/mod_perl_rules1.pl \
/home/httpd/perl/mod_perl_rules2.pl
```

The first command makes the files belong to group httpd, the second sets the proper execution and read permissions.

That's secure, assuming that you have a dedicated groupname for your server, of course.

Also remember that all the directories that lead to the script should be readable and executable by the server.

You can test mod_perl_rules1.pl from the command line, since it is essentially a regular Perl script.

```
% perl /home/httpd/perl/mod_perl_rules1.pl
```

You should see the following output:

```
mod_perl rules!
```

You cannot test the second script by executing it from the command line since it uses the mod_perl API which is available only when run from within the mod_perl server.

Make sure the server is running and issue these requests using your favorite browser:

http://localhost/perl/mod_perl_rules1.pl
http://localhost/perl/mod_perl_rules2.pl

In both cases you will see on the following response:

```
mod_perl rules!
```

If you see it−−congratulations! You have a working mod_perl server.

If you're using port 8080 instead of 80, you should use this number in the URL:

http://localhost:8080/perl/mod_perl_rules1.pl
http://localhost:8080/perl/mod_perl_rules2.pl

The localhost approach will work only if the browser is running on the same machine as the server. If not−−use the real server name for this test, for example:

http://your.server.name/perl/mod_perl_rules1.pl

If there is any problem please refer to the error_log file for the error reports.

Now it's a time to move your CGI scripts from /somewhere/cgi−bin directory to /home/httpd/perl/ and see them running much much faster, when requested from the newly configured base URL (/perl/). If you were accessing the script as /cgi−bin/test.pl, it will now be accessed from /perl/test.pl.

Some of your scripts might not work immediately and will require some minor tweaking or even a partial rewrite to work properly with mod_perl. Chances are that if you are not practicing sloppy programming, the scripts will work without any modifications at all.

If you have a problem with your scripts, a good approach is to replace Apache::Registry with Apache::PerlRun in httpd.conf, as the latter can execute really badly written scripts. Put the following configuration directives instead in httpd.conf and restart the server:

```
PerlModule Apache::PerlRun
<Location /perl>
SetHandler perl−script
PerlHandler Apache::PerlRun
Options ExecCGI
PerlSendHeader On
allow from all
</Location>
```

Now your scripts should work for sure, unless there is something in them mod_perl doesn't accept. We will discuss these nuances in future articles.

The "mod_perl rules" Apache Perl Module

mod_perl is about running both scripts and handlers. Although I have started to present mod_perl using scripts because it's easier if you have written CGI scripts before, the more advanced use of mod_perl is about writing handlers. But have no fear. As you will see in a moment, writing handlers is almost as easy as writing scripts.

To create a mod_perl handler module, all I have to do is to wrap the code I have used for the script into a handler subroutine, add a statement to return the status to the server when the subroutine has successfully completed, and append a package declaration at the top of the code.

Just as with scripts you can use either the CGI API you are probably used to (save the following as Rules1.pm):

```
package ModPerl::Rules1;
use Apache::Constants qw(:common);

sub handler{
print "Content-type: text/plain\r\n\r\n";
print "mod_perl rules!\n";
return OK;
}
1; # satisfy require()
```

or the Apache Perl API that allows you to interact more intimately with the Apache core by providing an API unavailable under regular Perl. Of course in the simple example that I show, using any of the approaches is fine, but when you need to use the API, this version of the code should be used (save as Rules2.pm):

```
package ModPerl::Rules2;
use Apache::Constants qw(:common);

sub handler{
my $r = shift;
$r->send_http_header('text/plain');
print "mod_perl rules!\n";
return OK;
}
1; # satisfy require()
```

Create a directory called ModPerl under one of the directories in @INC (e.g. /usr/lib/perl5/site_perl/5.005), and put Rules1.pm Rules2.pm into it, the files should include the code from the above examples.

To find out what the @INC directories are, execute:

```
% perl −le 'print join "\n", @INC'
```

On my machine it reports:

```
/usr/lib/perl5/5.6.1/i386−linux
/usr/lib/perl5/5.6.1
/usr/lib/perl5/site_perl/5.6.1/i386−linux
/usr/lib/perl5/site_perl/5.6.1
/usr/lib/perl5/site_perl
.
```

Now add the following snippet to httpd.conf to configure mod_perl to execute the ModPerl::Rules::handler subroutine whenever a request to mod_perl_rules1 is made:

```
PerlModule ModPerl::Rules1
<Location /mod_perl_rules1>
SetHandler perl−script
PerlHandler ModPerl::Rules1
</Location>
```

Now you can issue a request to:

mod_perl rules!

---

as the response.

To test the second module ModPerl::Rules2 add the same configuration, while replacing all 1's with 2's:

PerlModule ModPerl::Rules2 <Location /mod_perl_rules2> SetHandler perl−script PerlHandler ModPerl::Rules2 </Location> [code]

And to test use the URI:

http://localhost/mod_perl_rules2

Is This All I Need to Know About mod_perl?

Obviously the next question you'll ask is: "Is this all I need to know about mod_perl?".

The answer is: "Yes and No".

The Yes part:

Just like with Perl, you have to know very little about mod_perl to do really cool stuff. The presented setup allows you to run your visitor counters and guest book much faster and amaze your friends, usually without changing a single line of code.

The No part:

A 50 times improvement in guest book response times is great, but when you deploy a very heavy service with thousands of concurrent users, taking into account a high level competition between similar web services, a delay of a few milliseconds might cost you a customer and probably many of them.

Of course when you test a single script and you are the only user, you don't really care about squeezing yet another millisecond from response time, but it becomes a real issue when these milliseconds add up at the production site, with hundreds of users concurrently generating requests to various scripts on your site. Users aren't merciful nowadays−−if there is another even less fancier site that provides the same service but a little bit faster, chances are that they will go over there.

Testing your scripts on an unloaded machine can be very misleading, Everything might seem so perfect. But when you move them into a production machine, things don't behave as well as they did on your development box. Many times you just run out of memory on very busy services. You need to learn how to optimize your code to use less memory and how to make the memory shared.

Debugging is something people prefer not to talk about, since the process can be very tedious at times. Learning how to make the debugging process simpler and efficient is a must if you consider yourself a web programmer. This task is especially not so straightforward when debugging CGI scripts, and even more complicated with mod_perl. Unless you know how, and then it suddenly becomes easy.

mod_perl has many features unavailable under mod_cgi when working with databases. Among others the most important are persistent connections.

You have to know how to keep your service running non−stop and be able to recover fast if there are any problems.

Finally, the most important thing is the Apache−Perl API, which allows you to do anything with a received request, even intervene in every stage of request processing. This gives you great flexibility and lets you create things you couldn't dream about with plain mod_cgi.

There are many more things to learn about mod_perl and web programming in general. In future articles I'll talk in details about all these issues.

Acknowledgements

Many thanks to Eric Cholet for reviewing this article.

References

The Apache site's URL: http://www.apache.org/

The mod_perl site's URL: http://perl.apache.org/

CPAN is the Comprehensive Perl Archive Network. The Master site's URL is http://cpan.org/. CPAN is mirrored at more than one hundred sites around the world. (http://cpan.org/SITES.html)