Configuring Apache

Introduction

Apache is the foremost World-Wide Web server in use on the Internet. RedHat Linux supplies Apache as a package.

See Also

chmod (1), chown (1), kill (1), htpasswd (1), httpd (8), mount (8), rpm (8), ps (1), lynx (1)

The Task

In this exercise you will install and configure the Apache server.

You will need to be root for this exercise.

Installation

If you did not select the web components during installation, you should install the RedHat Apache package now. Issue the following commands:

```
# mount /dev/cdrom /mnt/cdrom
# rpm -ivvh /mnt/cdrom/Redhat/RPMS/apache-1.2.6-4.i386.rpm
```

(Note that the CD-ROM mount point and exact package name may vary for the system which you are using…)

The following commands will let you look at the system process list, start the Apache server and then see what new processes have been added to the system:

```
# ps ax
# /etc/rc.d/init.d/httpd restart
# ps ax
```

You can use the following command to ensure that you have installed Apache correctly:

```
# lynx http://localhost/
```

Configuration

The main configuration file for Apache is /etc/httpd/conf/httpd.conf. It contains lots of comments and so is very well documented.

Edit this file to make the following changes to Apache's configuration:

- change the MinSpareServers entry to 1
- set the MaxSpareServers value to 2
- set StartServers to 2
- set MaxClients to 15
- set the Timeout value to 10 seconds

This configuration is suitable only for a VERY lightly loaded system on a small(ish) LAN (but it will be OK for our current laboratory needs). These changes ensure that Apache's httpd server daemons do not consume too many resources (mainly RAM, swap space and CPU time).

You can test this configuration to ensure that you have edited the httpd.conf file appropriately:

```
# httpd -t
```

Configuring Apache

Once you have saved and tested the configuration file, you should tell httpd to reconfigure itself by using the following command:

```
# kill -HUP `cat /var/run/httpd.pid`
```

If you now look at the system process list, you should see that there are rather fewer httpd processes than there were before—the main httpd server will have terminated unneeded ones (according to its new configuration).

Changing A Server Response

Issue the following command:

```
# lynx localhost/xx
```

You will receive an error 404 alert along with a message telling you that “The requested URL /xx was not found on this server.”

This is not a very friendly way of telling the visitors to the server that something is not as they expected it to be! In addition, a person using a graphical browser would find the text-only message rather boring. It would be preferable to give a more friendly response, perhaps with an associated picture.

The html file returned by the server to the browser in response to error number 404 is specified in the /etc/httpd/conf/srm.conf file. Look in this file for the line:

```
#ErrorDocument 404 /missing.html
```

Remove the initial comment character (the ‘#’) and save the file.

You should now type the following into the file /home/httpd/html/missing.html:

```
<HTML>
<TITLE>Error 404</TITLE>
<BODY>
<P = CENTER>
<IMG SRC="../icons/broken.gif">
You have received an "ERROR 404" response.
This means that the URL you have just presented does not refer to a document
accessible to this server.
<IMG SRC="../icons/broken.gif">
</P>
<P>
Perhaps you should try the
<A HREF="index.html">home page.</A>
</P>
</BODY>
</HTML>
```

Once you have edited this file, ensure that its protections will allow it to be accessed by the server and then tell the server to reconfigure itself:

```
# chmod 644 /home/httpd/html/missing.html
# kill -HUP `cat /var/run/httpd.pid`
```

If you now try to access a missing page, your server will give a better(?) error message. Many of the other server messages can be tailored in much the same way.

Adding Access Controls To A Web Site

It is common to want to restrict access to some parts of a web site. Apache allows for this desire.

If you examine the file /etc/httpd/conf/srm.conf you will see the following line:

Configuring Apache

```
AccessFileName .htaccess
```

This causes the Apache server to look for a file called .htaccess in the directory containing the file currently being requested. If this file exists, the server will examine its contents for an access control list that defines the clients that are allowed to access the files contained in the directory.

Create the file /home/httpd/html/.htaccess and place the following inside it:

```
AuthType Basic
AuthUserFile /etc/httpd/conf/.htpasswd
AuthGroupFile /dev/null
AuthName "Access to this server is restricted..."
<Limit GET POST>
require user webuser
</Limit>
```

Ensure that this file has 644 protection.

Edit the file /etc/httpd/conf/access.conf. Look for the entry that establishes access permissions for the document root directory and modify the AllowOverride section:

```
<Directory /home/httpd/html>
[…snip…]
AllowOverride None
[…snip…]
</Directory>
```

to:

```
AllowOverride AuthConfig
```

You only need to modify a single line of this file…

You will also need to populate a password file as referenced by the AuthUserFile stanza of the above .htaccess file. Use the following command to create a user and give that user a password:

```
# htpasswd -c /etc/httpd/conf/.htpasswd webuser
```

Ensure that this file has 664 protection and is owned by user root, group root (use chmod and chown to ensure these requirements).

Once you have done this, you should try and access the web server again…you will see that it is only accessible to the user 'webuser' and then only if the correct password is presented.

Viewing The Apache Log Files

Try and access the web server via other usernames and passwords, to see how the server responds and also to build up a number of log entries.

Once you have done this, you will be able to see failed connection attempts by examining the file /var/log/httpd/error_log. You can also see a complete log of all files served at /var/log/httpd/access_log.

Setting Up A Simple Common Gateway Interface (CGI) Program

Apache needs no extra configuration to run simple CGI applications such as the one shown here:

Configuring Apache

You should obtain the file cgi-lib.pl from the instructor and place it in the directory /home/httpd/cgi-bin.

You should then create the following PERL program in the file /home/httpd/cgi-bin/guess.pl:

```
#!/usr/bin/perl -T
# the -T option does 'taint' checking

require './cgi-lib.pl';

$maxNum=10;
$thisScript='/cgi-bin/guess.pl';

print &PrintHeader;

&ReadParse(*input);
if ($input{'done'} eq 'done')
  {
  $val=$input{'val'};
  print '<TITLE>Your Result...</TITLE>';
  print '<BODY>';
  if ($val eq $input{'name'})
    {
    print "<H1>Good Guess!</H1>";
    }
  else
    {
    print "<H1>Bad Luck!</H1>";
    print "The number was $val. Better luck next time!";
    }
  print "<P><A HREF=\"$thisScript\">Try Again?</A></P>";
  print '<P><A HREF="/index.html">Go home</A></P>';
  print '</BODY>';
  }
else
  {
  $rv=int(rand($maxNum));
  print '<TITLE>Guess a number</TITLE>';
  print <<EOF

<BODY>
<H1>Guess a number....</H1>
<BR>
<FORM METHOD="POST" ACTION="$thisScript">
Guess a number between 0 and $maxNum:
<INPUT NAME="name">
<P></P>
<P></P>
<INPUT TYPE="HIDDEN" NAME="done" VALUE="done">
<INPUT TYPE="HIDDEN" NAME="val" VALUE="$rv">
<hr> <INPUT TYPE="RESET" VALUE="Clear">
<INPUT TYPE="SUBMIT" VALUE="Guess Now!">
</FORM>
</BODY>
EOF
  }
```

Once you have created this file, ensure that it has access mode 755 and then point your browser at the URL:

Configuring Apache

> http://localhost/cgi-bin/guess.pl

Good luck!