RunningMultipleApacheInstances

Running Multiple Apache Instances

This is a Recipe for configuring multiple https instances. While writing this recipe I've noticed that there are other pages in this wiki that also deal with this subject, either tangently or in more direct ways:

- ExtendingPrivilegeSeparation – run several httpd instances behind a reverse proxy, use high ports for the backend instances to improve security
- DifferentUserIDsUsingReverseProxy – run several https instances behind a reverse proxy, run each instance under an different unprivileged user

Some of the ideas in this recipe have already been advanced in these other wiki pages. However, the focus of this recipe is more on the operating system support (FreeBSD, Ubuntu) for this type of set up. The operating system usually provides facilities, such as rc(8) in BSD systems, or init(5) in Linux and other SysV systems, that help admins in launching processes after sucessfully booting the system.

This recipe, therefore complements those already in the wiki, and provides a step-by-step guide to configure such a setup in a BSD system (FreeBSD) and in a SysV-like system (Ubuntu). Hopefully it can be easily translated to other linux or *BSD flavours.

Note: The Ubuntu section is still a work in progress.

_
fernan

The Goal

Or: why do we need to run multiple apache instances in one host?

Although you can certainly have a single installation of Apache httpd, running a single instance, and still have different virtual hosts that can be accessed separately, sometimes following this easy path can lead you to a heavy and bloated web server.

If your virtual hosts have different requirements (e.g. mod_perl for one virtual host, mod_python for another, and maybe mod_php for a third virtual host), then your apache instance is almost surely a RAM eater. Remember that httpd will instantly spawn new child processes of the same instance as needed ... if your perl web app is being frequently accessed, why spawn several instances of an httpd process that is also loaded with mod_python, mod_php and an assortment of other modules that, at least from the standpoint of the perl web app, are useless?

The answer to this, is simple: have separate lighter configurations for your mod_perl, mod_python and mod_php apps, listening on different ports (e.g. 81, 82, 83), and have a reverse proxy instance of Apache with virtual hosts configured to pass requests to the corresponding apache instances.

In this recipe, I'll briefly explain how to do something along these lines:
Example: running different web applications listening in different ports, and using a reverse proxy that forwards incoming requests (e.g. blog.company.com, svn.company.com, www.company.com) to the corresponding https instances.

HowTo

How do we configure apache so that we're able to start/stop/restart individual instances?

Because apache allows you to load modules dynamically, most of the time, a single installation of apache can serve the needs of multiple instances, and every instance may be configured to run different sets of modules. As an example, you can run a very lightweight proxy module, a mod_perl instance to run your perl web applications, a mod_python instance with DAV to run Subversion, a separate instance running PHP, etc, all from a single local installation of Apache. Unless you have a requirement for a custom built instance (e.g. tweaking compile options differently for different instances), you don't really need to compile and install httpd to different places (prefixes), as many online guides suggest.

In FreeBSD, a clever port maintainer had set up a number of rc scripts that simplify the task of launching all configured instances after booting the system. So if you're using FreeBSD as your server OS, then you're lucky, as the configuration is painless.

In this recipe, therefore, I'll proceed to explain the FreeBSD approach first, and then I'll try to figure out how to do something similar in Linux (Ubuntu).

The FreeBSD approach

1. Install the www/apache22 port

As root, do:

```
cd /usr/ports/www/apache22
# configure the port: select compile options and modules
make config
# now make and install
make clean && make install clean
```

This will fetch the sources for Apache (currently 2.2.17), extract them, and run configure, make and finally install apache-22 on your system. As mentioned before, note that only one local installation is required to run multiple instances of apache. Also note that if you configure the port (*make config*) to enable mod_perl, mod_python (or any other dynamically loaded module that does not come with the standard apache distribution), then the FreeBSD ports system will 'Do The Right Thing(TM)' and will fetch, extract, configure, make and install the code for the corresponding modules. Easy, huh?

Note also, that there are other alternative ports in FreeBSD that will install different flavours of Apache https:

- www/apache22-event-mpm
- www/apache22-peruser-mpm
- www/apache22-worker-mpm
- etc.

Choose the port that best fits your needs, and proceed to the next step.

2. Create your apache configurations

In FreeBSD, the configuration files for the recently installed port live in '/usr/local/etc/apache22'. There you will find a sample 'httpd.conf' file. This file may include directives found in other conf files located in a directory called 'extra', at the same location. Please read them, and familiarize yourself with their contents. If you've configured Apache before, you'll find your way around easily.

Now, let's create a number of separate apache configurations. Essentially this means creating different httpd.conf files, for example:

```
cp httpd.conf httpd-proxy.conf
cp httpd.conf httpd-perl.conf
cp httpd.conf httpd-python.conf
cp httpd.conf httpd-php.conf
...
```

Now edit these files, and tweak the different configurations according to your needs, e.g.

```
# httpd-proxy.conf
Listen 80
ErrorLog /var/log/httpd-proxy-error.log
LoadModule proxy_module                 libexec/apache22/mod_proxy.so
LoadModule proxy_http_module            libexec/apache22/mod_proxy_http.so

# httpd-perl.conf
Listen 81
ErrorLog /var/log/httpd-perl-error.log
LoadModule perl_module                  libexec/apache22/mod_perl.so

# httpd-python.conf
Listen 82
ErrorLog /var/log/httpd-python-error.log
LoadModule python_module                libexec/apache22/mod_python.so

# httpd-php.conf
Listen 83
ErrorLog /var/log/httpd-php-error.log
LoadModule php5_module                        libexec/apache22/libphp5.so
```

Now, we also need to configure the virtual hosts in the proxy instance, so that whenever a request comes for the Subversion DAV server, it is passed onto your 'python-dav' httpd instance, whereas requests for your wordpress blog are passed to your 'php' instance. Let's edit 'httpd-proxy.conf' again:

```
# httpd-proxy.conf
NameVirtualHost *:80
<VirtualHost *:80>
 DocumentRoot /www/wordpress
 ServerName blog.company.com
 ProxyPass / http://localhost:83/
 ProxyPassReverse / http://localhost:83/
 [... additional directives here ... ]
</VirtualHost>
<VirtualHost *:80>
 DocumentRoot /www/svn
 ServerName svn.company.com
 ProxyPass / http://localhost:82/
 ProxyPassReverse / http://localhost:82/
 [... additional directives here ... ]
</VirtualHost>
# you get the idea ...
# you might need to use the "ProxyPreserveHost On" directive, depending on your configuration
```

So we're finished with the configuration, and now we need to launch all the instances of httpd, and test that everything is working as expected. Of course you can do this using 'apachectl', e.g.

```
/usr/local/sbin/apachectl -f /usr/local/etc/apache22/proxy-httpd.conf configtest
/usr/local/sbin/apachectl -f /usr/local/etc/apache22/proxy-httpd.conf start
/usr/local/sbin/apachectl -f /usr/local/etc/apache22/perl-httpd.conf configtest
/usr/local/sbin/apachectl -f /usr/local/etc/apache22/perl-httpd.conf start
# and so on ...
```

But for production operation, it is safer to rely on the FreeBSD rc scripts. These will make sure that the instances are launched at boot time. Our last stop in this FreeBSD recipe, is therefore:

3. Edit /etc/rc.conf

The '/etc/rc.conf' in FreeBSD is the master file containing the system configuration information. This file is read after booting the kernel, and serves to launch services, daemons, set up network interfaces, etc. For our recipe we will be enabling the apache server, listing the available instances (*profiles*), their configuration files, and telling FreeBSD which of these need to be run (*enabled*) after booting the system.

```
# /etc/rc.conf
apache22_enable="YES"
apache22_profiles="proxy perl python php"

# the apache proxy instance
apache22_proxy_configfile="/usr/local/etc/apache22/httpd-proxy.conf"
apache22_proxy_enable="YES"

# the apache perl instance
apache22_perl_configfile="/usr/local/etc/apache22/httpd-perl.conf"
apache22_perl_enable="YES"

# the apache python instance
apache22_python_configfile="/usr/local/etc/apache22/httpd-python.conf"
apache22_python_enable="YES"

# the apache php instance
apache22_php_configfile="/usr/local/etc/apache22/httpd-php.conf"
apache22_php_enable="YES"
```

When these *profiles* are configured in /etc/rc.conf, and *enabled*, they will be started after successfully booting the system. If you want to declare a profile but you only want to start the corresponding httpd instance manually, you can just edit '/etc/rc.conf' and say, e.g. :

```
# the apache php instance
apache22_php_configfile="/usr/local/etc/apache22/httpd-php.conf"
apache22_php_enable="NO"
```

Later, you can start/stop any httpd instance manually using just the profile name (proxy, perl, python, php), like this:

```
/usr/local/etc/rc.d/apache22 start php
/usr/local/etc/rc.d/apache22 stop perl
...
```

Doing it in Ubuntu or Debian

1. Install the apache2 package

In Ubuntu, you need to install not only the apache2 package, but also, the corresponding libapache2-* packages for the modules you need in your apache instances.

```
sudo apt-get install apache2
sudo apt-get install libapache2-mod-perl2
sudo apt-get install libapache2-mod-python
sudo apt-get install libapache2-mod-php5
...
```

2. Create your apache configurations

In Ubuntu, the configuration files for your recently installed webserver are under '/etc/apache2'.
[writing in progress ...]

There is some Debian/Ubuntu specific information in /usr/share/doc/apache2/README.Debian.gz after installation.

3. Configure the system to launch the apache instances after boot

The Ubuntu/Debian init scripts (e.g. /etc/init.d/apache2) have been updated to support multiple instances of (e.g. multiple configurations, named /etc /apache2-$SUFFIX).

Documentation can be found in /usr/share/doc/apache2/README.multiple-instances