Tutorial 1 - Compiling an Apache web server

What are we doing?

We're compiling an Apache web server for a test system.

Why are we doing this?

In professional use of the web server it's very often the case that special requirements (security, additional debugging messages, special features from a new patch, etc.) force you to leave behind the distribution packages and quickly create some binaries on your own. In this case it's important for the infrastructure to be prepared and to already have experience compiling and running your own binaries on production systems. It's also easier to work with self-compiled Apache in a laboratory-like setup, which is also beneficial in terms of debugging.

Step 1: Preparing the directory tree for the source code

It's not all that important where the source code is located. The following is a recommendation based on the File Hierarchy Standard. The FHS defines the path structure of a Unix system; the structure for all stored files.

```
$> sudo mkdir /usr/src/apache
$> sudo chown `whoami` /usr/src/apache
$> cd /usr/src/apache
```

Step 2: Meeting the requirements for apr and apr-util

Since the release of version 2.4, the Apache web server comes without two important libraries that used to be part of the distribution. We now have to install `apr` and `apr-util` ourselves before being able to compile Apache. `apr` is the Apache Portable Runtime library. It adds additional features to the normal set of C libraries typically needed by server software. They include features for managing hash tables and arrays. These libraries aren't used by the Apache web server alone, but also by other Apache Software Foundation projects, which is why they were removed from Apache's source code. Like `apr`, `apr-util` is part of the Portable Runtime libraries supplemented by `apr-util`.

Let's start with *apr* and download the package.

```
$> wget http://mirror.switch.ch/mirror/apache/dist/apr/apr-1.5.2.tar.bz2
```

We'll now download the checksum of the source code file from Apache. Unfortunately, *www.apache.org* only offers an md5 checksum for `apr`. We'll verify it anyway. For better security we'll be using a secure connection for downloading. Without https this verification doesn't make much sense. Both files, the source code and the small checksum file, should be placed together in `/usr/src/apache`. We can now verify the checksum:

```
$> wget https://www.apache.org/dist/apr/apr-1.5.2.tar.bz2.md5
$> md5sum --check apr-1.5.2.tar.bz2.md5
apr-1.5.2.tar.bz2: OK
```

The test should not result in any problems, *OK*. We can now continue with unpacking, pre-configuring and compiling *apr*.

```
$> tar xvjf apr-1.5.2.tar.bz2
$> cd apr-1.5.2
$> ./configure --prefix=/usr/local/apr/
```

After unpacking, we now change to the new directory containing the source code and start *configure*. This configures the compiler. We specify the installation path and *configure* gathers a variety of information and settings about our system. The configure command frequently complains about missing components. One thing is certain: Without a working compiler we will be unable to compile and it's configure's task to check whether everything is assembled correctly.

Things typically missing:

- build-essential
- binutils
- gcc

These are the package names on Debian-based distributions. The packages may have different names elsewhere. The absence of these files can be easily rectified by re-installing them using the utilities from your own distribution. Afterwards, run *configure* again, perhaps re-install something again and eventually the script will run successfully.

Once it runs without a problem, we can assume that the time for compiling has come.

```
$> make
```

This takes a moment after which we get the compiled *apr*, which we promptly install.

```
$> sudo make install
```

Once this is successful, we'll do the same with *apr-util*.

```
$> cd /usr/src/apache
$> wget http://mirror.switch.ch/mirror/apache/dist/apr/apr-util-1.5.4.tar.bz2
$> wget https://www.apache.org/dist/apr/apr-util-1.5.4.tar.bz2.md5
$> md5sum --check apr-util-1.5.4.tar.bz2.md5
apr-util-1.5.4.tar.bz2: OK
$> tar xvjf apr-util-1.5.4.tar.bz2
$> cd apr-util-1.5.4
$> ./configure --prefix=/usr/local/apr/ --with-apr=/usr/local/apr/
$> make
$> sudo make install
```

Once this works in both cases we're ready for the web server itself.

Step 3: Downloading the source code and verifying the checksum

We'll now download the program code from the internet. This can be done by downloading it directly from Apache in a browser or, to save the Apache Project's bandwidth, by using wget to get it from a mirror.

```
$> cd /usr/src/apache
$> wget http://mirror.switch.ch/mirror/apache/dist//httpd/httpd-2.4.17.tar.bz2
```

The compressed source code is approximately 5 MB in size.

We'll now download the checksum of the source code file from Apache. At least it's available as a *sha1 checksum*. We'll again be using a secure connection for better security. Without https this verification doesn't make much sense.

```
$> wget https://www.apache.org/dist/httpd/httpd-2.4.17.tar.bz2.sha1
$> sha1sum --check httpd-2.4.17.tar.bz2.sha1
httpd-2.4.17.tar.bz2: OK
```

Step 4: Unpacking and configuring the compiler

After verification we can unpack the package.

```
$> tar xvjf httpd-2.4.17.tar.bz2
```

This results in approximately 38 MB.

We now enter the directory and configure the compiler with our entries and with information about our system. Unlike *apr*, our entries are very extensive.

```
$> cd httpd-2.4.17
$> ./configure --prefix=/opt/apache-2.4.17  --with-apr=/usr/local/apr/bin/apr-1-config \
   --with-apr-util=/usr/local/apr/bin/apu-1-config \
   --enable-mpms-shared=event \
   --enable-mods-shared=all \
   --enable-nonportable-atomics=yes
```

This is where we define the target directory for the future Apache web server, again compiling in compliance with the *FHS*. Following this, there are two options for linking the two libraries installed as a precondition. We use `--enable-mpms-shared` to select a process model for the server. Simply put, this is like an engine type: gasoline (petrol) or diesel. In our case, `event`, `worker`, `prefork` and a few experimental engines are available. In this case we'll take the `event` model, which is the new standard in 2.4 and has significantly better performance over the other architectures. In the 2.0 and 2.2 version lines there was significantly more to consider besides performance, but this set of problems has been significantly defused since 2.4 and it's best for us to continue with `event`. More information about the different process models (*MPMs*) is available from the Apache Project.

We then define that we want all (*all*) modules to be compiled. Of note here is that *all* does not really mean all. For historical reasons *all* means only all of the core modules, of which there are quite a few. The *shared* keyword indicates that we would like to have the modules compiled separately in order to then be able to link them as optional modules. And lastly, `enable-nonportable-atomics` is a compiler flag which instructs the compiler to use special options which are available only on modern x86 processors and have a favorable impact on performance.

When executing the *configure* command for the web server, it may be necessary to install additional packages. These may include:

- libpcre3-dev
- libssl-dev
- zlibc
- zlib1g-dev

Depending on distribution, one or more of these packages may have different names.

When compiling it is often the case that some components are missing. Installing packages is at times more difficult than in our case and versions may be incompatible in the worst case. There's often a solution to the problem on the internet, but now and again you'll have to dive really deep into the system to get to the root of the difficulties. This should not be an issue in our simple case.

Step 5: Compiling

Once *configure* is completed, we are ready for the compiler. Nothing should go wrong any longer at this point.

```
$> make
```

This takes some time and 38 MB becomes just under 100 MB.

Step 6: Installing

When compiling is successful, we then install the Apache web server we built ourselves. Installation must be performed by the super user. But right afterwards we'll see how we can again take ownership of the web server. This is much more practical for a test system.

```
$> sudo make install
```

Installation may also take some time.

```
$> sudo chown -R `whoami` /opt/apache-2.4.17
```

And now for a trick: If you work professionally with Apache then you often have several different versions on the test server. Different versions, different patches, other modules, etc. result in tedious and long pathnames with version numbers and other descriptions. To ease things, I create a soft link from `/apache` to the current Apache web server when I switch to a new

version. Care must be given that we and not the root user are the owners of the soft link (this is important in configuring the server).

```
$> sudo ln -s /opt/apache-2.4.17 /apache
$> sudo chown `whoami` --no-dereference /apache
$> cd /apache
```

Our web server now has a pathname clearly describing it by version number. We will however simply use `/apache` for access. This makes work easier.

Step 7: Starting

Now let's see if our server will start up. For the moment, this again has to be done by the super user:

```
$> sudo ./bin/httpd -X
```

Another trick for test operation: Apache is actually a daemon running as a background process. However, for simple tests this can be quite bothersome, because we have to continually start, stop, reload and otherwise manipulate the daemon. The -X option tells Apache that it can do without the daemon and start as a single process/thread in the foreground. This also simplifies the work.

There is likely to be a warning when starting:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set
```

This is unimportant and we can ignore the warning for the time being.

Step 8: Trying it out

The engine is running. But is it also working? Time for the function test: We access Apache by entering the following URL in our browser:

http://127.0.0.1

We then expect the following:

Apache shows the first signs of life in the browser.

Fantastic! Goal achieved: The self-compiled Apache is running.

Return to the shell and stop the server via CTRL-C.

Step 9 (Bonus): Inspecting the binaries and the modules

Before completing the tutorial, we'd like to take a closer look at the server. Let's open the engine compartment and take a peek inside. We can get information about our binary as follows:

```
$> sudo ./bin/httpd -V


Server version: Apache/2.4.17 (Unix)
Server built:   Oct 16 2015 21:09:49
Server's Module Magic Number: 20120211:47
Server loaded:  APR 1.5.2, APR-UTIL 1.5.4
Compiled using: APR 1.5.2, APR-UTIL 1.5.4
Architecture:   64-bit
Server MPM:     event
  threaded:     yes (fixed thread count)
    forked:     yes (variable process count)
Server compiled with....
 -D APR_HAS_SENDFILE
 -D APR_HAS_MMAP
 -D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
 -D APR_USE_SYSVSEM_SERIALIZE
 -D APR_USE_PTHREAD_SERIALIZE
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D APR_HAS_OTHER_CHILD
 -D AP_HAVE_RELIABLE_PIPED_LOGS
 -D DYNAMIC_MODULE_LIMIT=256
 -D HTTPD_ROOT="/opt/apache-2.4.17"
 -D SUEXEC_BIN="/opt/apache-2.4.17/bin/suexec"
 -D DEFAULT_PIDLOG="logs/httpd.pid"
 -D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
 -D DEFAULT_ERRORLOG="logs/error_log"
```

```
  -D AP_TYPES_CONFIG_FILE="conf/mime.types"
  -D SERVER_CONFIG_FILE="conf/httpd.conf"
```

Because we specified the version when we compiled, `apr` is mentioned and the `event` *MPM* appears further below. Incidentally, at the very bottom we see a reference to the web server's default configuration file and a bit above this the path we can use to find the default *error_log*.

You can however get even more information from the system and inquire about the modules compiled firmly into the server.

```
$> sudo ./bin/httpd -l
```

```
Compiled in modules:
  core.c
  mod_so.c
  http_core.c
```

This and the information above can be helpful for debugging and useful when submitting bug reports. These are typically the first questions that the developer asks.

The binary itself (`/apache/bin/httpd`) is approx. 2.0 MB in size and the list of modules appears as follows:

```
$> ls -lh modules
```

```
total 8.8M
-rw-r--r-- 1 myuser dune73  14K Oct 16 21:09 httpd.exp
-rwxr-xr-x 1 myuser root    36K Oct 16 21:16 mod_access_compat.so
-rwxr-xr-x 1 myuser root    34K Oct 16 21:17 mod_actions.so
-rwxr-xr-x 1 myuser root    49K Oct 16 21:17 mod_alias.so
-rwxr-xr-x 1 myuser root    31K Oct 16 21:17 mod_allowmethods.so
-rwxr-xr-x 1 myuser root    30K Oct 16 21:17 mod_asis.so
-rwxr-xr-x 1 myuser root    47K Oct 16 21:16 mod_auth_basic.so
-rwxr-xr-x 1 myuser root   102K Oct 16 21:16 mod_auth_digest.so
-rwxr-xr-x 1 myuser root    79K Oct 16 21:16 mod_auth_form.so
-rwxr-xr-x 1 myuser root    30K Oct 16 21:16 mod_authn_anon.so
-rwxr-xr-x 1 myuser root    39K Oct 16 21:16 mod_authn_core.so
-rwxr-xr-x 1 myuser root    43K Oct 16 21:16 mod_authn_dbd.so
-rwxr-xr-x 1 myuser root    33K Oct 16 21:16 mod_authn_dbm.so
-rwxr-xr-x 1 myuser root    33K Oct 16 21:16 mod_authn_file.so
-rwxr-xr-x 1 myuser root    54K Oct 16 21:16 mod_authn_socache.so
-rwxr-xr-x 1 myuser root    70K Oct 16 21:16 mod_authz_core.so
-rwxr-xr-x 1 myuser root    46K Oct 16 21:16 mod_authz_dbd.so
-rwxr-xr-x 1 myuser root    37K Oct 16 21:16 mod_authz_dbm.so
-rwxr-xr-x 1 myuser root    41K Oct 16 21:16 mod_authz_groupfile.so
-rwxr-xr-x 1 myuser root    37K Oct 16 21:16 mod_authz_host.so
-rwxr-xr-x 1 myuser root    31K Oct 16 21:16 mod_authz_owner.so
-rwxr-xr-x 1 myuser root    31K Oct 16 21:16 mod_authz_user.so
-rwxr-xr-x 1 myuser root   129K Oct 16 21:17 mod_autoindex.so
-rwxr-xr-x 1 myuser root    35K Oct 16 21:17 mod_buffer.so
-rwxr-xr-x 1 myuser root   103K Oct 16 21:17 mod_cache_disk.so
-rwxr-xr-x 1 myuser root   229K Oct 16 21:17 mod_cache.so
-rwxr-xr-x 1 myuser root   108K Oct 16 21:17 mod_cache_socache.so
-rwxr-xr-x 1 myuser root   118K Oct 16 21:17 mod_cgid.so
-rwxr-xr-x 1 myuser root    68K Oct 16 21:17 mod_charset_lite.so
-rwxr-xr-x 1 myuser root    33K Oct 16 21:17 mod_data.so
-rwxr-xr-x 1 myuser root   221K Oct 16 21:17 mod_dav_fs.so
-rwxr-xr-x 1 myuser root    83K Oct 16 21:17 mod_dav_lock.so
-rwxr-xr-x 1 myuser root   395K Oct 16 21:17 mod_dav.so
-rwxr-xr-x 1 myuser root    71K Oct 16 21:17 mod_dbd.so
-rwxr-xr-x 1 myuser root   100K Oct 16 21:17 mod_deflate.so
-rwxr-xr-x 1 myuser root    36K Oct 16 21:17 mod_dialup.so
-rwxr-xr-x 1 myuser root    37K Oct 16 21:17 mod_dir.so
-rwxr-xr-x 1 myuser root    33K Oct 16 21:17 mod_dumpio.so
-rwxr-xr-x 1 myuser root    34K Oct 16 21:17 mod_echo.so
-rwxr-xr-x 1 myuser root    32K Oct 16 21:17 mod_env.so
-rwxr-xr-x 1 myuser root    44K Oct 16 21:17 mod_expires.so
-rwxr-xr-x 1 myuser root    74K Oct 16 21:17 mod_ext_filter.so
```

```
-rwxr-xr-x 1 myuser root    42K Oct 16 21:17 mod_file_cache.so
-rwxr-xr-x 1 myuser root    62K Oct 16 21:17 mod_filter.so
-rwxr-xr-x 1 myuser root    73K Oct 16 21:17 mod_headers.so
-rwxr-xr-x 1 myuser root    30K Oct 16 21:17 mod_heartbeat.so
-rwxr-xr-x 1 myuser root    79K Oct 16 21:17 mod_heartmonitor.so
-rwxr-xr-x 1 myuser root   163K Oct 16 21:17 mod_include.so
-rwxr-xr-x 1 myuser root    85K Oct 16 21:17 mod_info.so
-rwxr-xr-x 1 myuser root    35K Oct 16 21:17 mod_lbmethod_bybusyness.so
-rwxr-xr-x 1 myuser root    35K Oct 16 21:17 mod_lbmethod_byrequests.so
-rwxr-xr-x 1 myuser root    35K Oct 16 21:17 mod_lbmethod_bytraffic.so
-rwxr-xr-x 1 myuser root    52K Oct 16 21:17 mod_lbmethod_heartbeat.so
-rwxr-xr-x 1 myuser root   103K Oct 16 21:17 mod_log_config.so
-rwxr-xr-x 1 myuser root    43K Oct 16 21:17 mod_log_debug.so
-rwxr-xr-x 1 myuser root    37K Oct 16 21:17 mod_log_forensic.so
-rwxr-xr-x 1 myuser root    35K Oct 16 21:17 mod_logio.so
-rwxr-xr-x 1 myuser root   467K Oct 16 21:17 mod_lua.so
-rwxr-xr-x 1 myuser root    56K Oct 16 21:17 mod_macro.so
-rwxr-xr-x 1 myuser root    88K Oct 16 21:17 mod_mime_magic.so
-rwxr-xr-x 1 myuser root    60K Oct 16 21:17 mod_mime.so
-rwxr-xr-x 1 myuser root   184K Oct 16 21:16 mod_mpm_event.so
-rwxr-xr-x 1 myuser root   136K Oct 16 21:16 mod_mpm_worker.so
-rwxr-xr-x 1 myuser root   117K Oct 16 21:17 mod_negotiation.so
-rwxr-xr-x 1 myuser root   198K Oct 16 21:17 mod_proxy_ajp.so
-rwxr-xr-x 1 myuser root   139K Oct 16 21:17 mod_proxy_balancer.so
-rwxr-xr-x 1 myuser root    59K Oct 16 21:17 mod_proxy_connect.so
-rwxr-xr-x 1 myuser root    40K Oct 16 21:17 mod_proxy_express.so
-rwxr-xr-x 1 myuser root    77K Oct 16 21:17 mod_proxy_fcgi.so
-rwxr-xr-x 1 myuser root    40K Oct 16 21:17 mod_proxy_fdpass.so
-rwxr-xr-x 1 myuser root   131K Oct 16 21:17 mod_proxy_ftp.so
-rwxr-xr-x 1 myuser root   114K Oct 16 21:17 mod_proxy_html.so
-rwxr-xr-x 1 myuser root   121K Oct 16 21:17 mod_proxy_http.so
-rwxr-xr-x 1 myuser root    66K Oct 16 21:17 mod_proxy_scgi.so
-rwxr-xr-x 1 myuser root   357K Oct 16 21:17 mod_proxy.so
-rwxr-xr-x 1 myuser root    59K Oct 16 21:17 mod_proxy_wstunnel.so
-rwxr-xr-x 1 myuser root    33K Oct 16 21:17 mod_ratelimit.so
-rwxr-xr-x 1 myuser root    34K Oct 16 21:17 mod_reflector.so
-rwxr-xr-x 1 myuser root    41K Oct 16 21:17 mod_remoteip.so
-rwxr-xr-x 1 myuser root    48K Oct 16 21:17 mod_reqtimeout.so
-rwxr-xr-x 1 myuser root    40K Oct 16 21:17 mod_request.so
-rwxr-xr-x 1 myuser root   210K Oct 16 21:17 mod_rewrite.so
-rwxr-xr-x 1 myuser root   144K Oct 16 21:17 mod_sed.so
-rwxr-xr-x 1 myuser root    35K Oct 16 21:17 mod_session_cookie.so
-rwxr-xr-x 1 myuser root    53K Oct 16 21:17 mod_session_dbd.so
-rwxr-xr-x 1 myuser root    61K Oct 16 21:17 mod_session.so
-rwxr-xr-x 1 myuser root    47K Oct 16 21:17 mod_setenvif.so
-rwxr-xr-x 1 myuser root    32K Oct 16 21:17 mod_slotmem_plain.so
-rwxr-xr-x 1 myuser root    59K Oct 16 21:17 mod_slotmem_shm.so
-rwxr-xr-x 1 myuser root    52K Oct 16 21:17 mod_socache_dbm.so
-rwxr-xr-x 1 myuser root    40K Oct 16 21:17 mod_socache_memcache.so
-rwxr-xr-x 1 myuser root    82K Oct 16 21:17 mod_socache_shmcb.so
-rwxr-xr-x 1 myuser root    43K Oct 16 21:17 mod_speling.so
-rwxr-xr-x 1 myuser root   897K Oct 16 21:17 mod_ssl.so
-rwxr-xr-x 1 myuser root    80K Oct 16 21:17 mod_status.so
-rwxr-xr-x 1 myuser root    48K Oct 16 21:17 mod_substitute.so
-rwxr-xr-x 1 myuser root    35K Oct 16 21:17 mod_unique_id.so
-rwxr-xr-x 1 myuser root    37K Oct 16 21:17 mod_unixd.so
-rwxr-xr-x 1 myuser root    34K Oct 16 21:17 mod_userdir.so
-rwxr-xr-x 1 myuser root    44K Oct 16 21:17 mod_usertrack.so
-rwxr-xr-x 1 myuser root    27K Oct 16 21:17 mod_version.so
-rwxr-xr-x 1 myuser root    40K Oct 16 21:17 mod_vhost_alias.so
-rwxr-xr-x 1 myuser root    54K Oct 16 21:17 mod_watchdog.so
-rwxr-xr-x 1 myuser root    69K Oct 16 21:17 mod_xml2enc.so
```

These are all of the modules distributed along with the server by Apache and we are well aware that we selected the *all* option for the modules to compile. Additional modules are available from third parties. We don't need all of these modules, but there are some you'll almost always want to have: They are ready to be included.

References

- Apache: http://httpd.apache.org
- File Hierarchy Standard: http://www.pathname.com/fhs/
- Apache ./configure documentation: http://httpd.apache.org/docs/trunk/programs/configure.html

License / Copying / Further use