Apache Web Server 2.4 – 10 Must-know Configuration Features

Rainer Jung
kippdata informationstechnolgie GmbH

Introduction

- Rainer Jung, kippdata GmbH
- Committer and PMC member for Apache Tomcat, the Apache httpd server and Apache JMeter
- Apache Software Foundation (ASF) member
- Doing lots of performance analysis and other troubleshooting
- Providing support for the Apache Web Server, Apache Tomcat and other web infrastructure

Disclaimer

- Our approach:
  - We will show some nice little tricks that are still not well known
  - We will not describe best practices for a basic configuration
  - None of these need 3rd-party modules

The list

- "Define" – using variables
- mod_macro – using templates
- Per module log levels
- Improved access log timestamps and more access log fields
- Correlation ID forwarding
- mod_log_debug – Custom log lines
- "<If EXPRESSION>" – conditional configuration
- "LocationMatch" and other *Match: named backreferences
- mod_remoteip: getting the client IP address right
- mod_ssl: "SSLOpenSSLConfCmd"

- Number 1: “Define” – using global config variables

```
Define STAGE dev
ProxyPass /app/ https://services-${STAGE}/app/
```

  - “Define” sets variables, “${...}” references them
  - You can also use “${}” to reference unix environment variables
    - But those can not be changed via restart or graceful
    - Define'd variables can be changed and activated with graceful!
  - Example use case: factor out varying names and sizings for different stages by replacing them with variables and setting the variables in a separate Include'd file.

„Define“ – using variables

- “UnDefine VAR” – remove variable “VAR”
- “<IfDefine VAR>...</IfDefine>” – Conditional configuration, active if “VAR” is defined (independent of the value of “VAR”
- Negation: <IfDefine !VAR>
- You can also define such a variable with the startup flag “-D VAR”, but you can not give it a value with “-D”.

- Example for combining “-D” and “Define”:
  - Control running config by including or not including the flag “-D TEST” on startup

```
<IfDefine TEST>
  Define servername test.example.com
</IfDefine>
<IfDefine !TEST>
  Define servername www.example.com
  Define SSL
</IfDefine>
```

- Number 2: mod_macro - using templates
  - A macro is a config snippet with parameters
  - One defines the macro with a "<Macro>" block:

```
<Macro VHost $servername $htdocs $logname>
  ServerName $servername
  ServerAdmin webmaster@example.org
  DocumentRoot $htdocs
  ErrorLog logs/error-$logname.log
  CustomLog logs/access-$logname.log
</Macro>
```

    - Here "VHost" is the name of the macro and "$servername", "$htdocs" and "$logname" are its parameters

- One can then call the macro with "Use" with varying parameters multiple times

```
<VirtualHost _default_:443>
  Use VHost www.example.org htdocs-www www
  ...
</VirtualHost>
<VirtualHost _default_:443>
  Use VHost sales.example.com htdocs-sales sales
  ...
</VirtualHost>
```

„Define“ versus mod_macro

- “Define” is especially useful if the same value should be used in various places in the config. Define once, use everywhere.
- “Define” is useful if you want to keep a list of parameters of the config in one place like an Include'd config file (consisting only of “Define” directives)
- A macro is useful if your configuration contains complex blocks that repeat with slight parameter changes

- Number 3: Per module log levels
  - There are fine-grained log levels for the error log: …, info, debug, **trace1, …, trace8**
  - No we can make one specific module (here: mod_rewrite) log very verbosely:

```
LogLevel info rewrite:trace8
```

  - You can narrow down the verbosity to virtual hosts and even directory context.
    - Note that log level changes in directory context get effective only after a request has been parsed, so that the server actually knows the URI and eventually directory.

- Number 4: Improved access log timestamps and more access log fields
  - By default time stamps in access logs have seconds granularity. This often is no longer appropriate, we need finer measurements
    - The "%t" pattern used to include time stamps in the access log allows for variations of the form "%{FORMAT}t
    - I suggest something like

```
[%{%d/%b/%Y:%T}t.%{usec_frac}t %{%z}t]
```

    - That is similar to the default timestamp, except it adds fractional microseconds!
    - The other %-patterns are "strftime()" patterns.

- More on access log timestamps
  - When combining an httpd reverse proxy with a Tomcat back end, using "%t" in access logs for both components:
    - httpd logs the timestamp of request **start**
    - Tomcat logs the timestamp of request **end**
  - You can change that in httpd 2.4 (or Tomcat) by using an explicit "begin:" or "end:" token prefix in the pattern:

```
[%{end:%d/%b/%Y:%T}t.%{end:usec_frac}t %{end:%z}t]
```

- Other useful access log fields
  - Not included by default but consider adding them
  - "%D": request duration in microseconds (!)
    - Tomcat: %D is millisecods
  - "%L": correlation ID. A unique ID that will also get logged for any error log line. You can use it to correlate error log lines with access log lines (without uncertainty).
    - Activate mod_unique_id for better %L values
  - "%{begin:usec}t %{end:usec}t": unix microsecond request start and response end time. Useful for easier calculations.

- Even more useful access log fields
  - "%P:%{tid}P": Process ID and thread ID
  - "%k %X": Connection keep-alive counter and final connection state

- Number 5: Correlation ID forwarding
  - We want to correlate log information between an httpd reverse proxy and the backends
  - Idea: each request should get a unique ID. Log the ID in the web server and forward it as a custom header to the next hop (back end).
  - How to generate a unique request ID in the web server? Easy:
    - Simply load mod_unique_id.
    - And add "%{UNIQUE_ID}e" to the access log format

Correlation ID forwarding

- How to forward the correlation ID?
  - We add a custom header using mod_headers:

```
RequestHeader set "X-Request-ID" "%{UNIQUE_ID}e"
```

  - What to do on the back end?
    - Add "%{X-Request-ID}i" to the back end access log format
    - Read that header via a servlet filter and pass it to your log framework, eg. via Log4J(2) MDC to automagically include it in every log line! That can be done by generic code.
  - Side remark: you can also use any string expression generated by the expression parser as a header value

- Number 6: mod_log_debug – Custom log lines
  - The module mod_log_debug provides "LogMessage"

```
LogLevel log_debug:info
...
<Location /honeypot>
  LogMessage "Honeypot accessed"
</Location>
```

  - You can choose the hook (specific part of request execution in the server) during which the logging should happen. The special hook name "all" logs in every hook:

```
LogMessage "Honeypot accessed" hook=all
```

- Using variables with mod_log_debug

```
LogMessage "Honeypot accessed by %{REMOTE_ADDR}"
```

- Using more complex variable expressions

```
RewriteCond %{HTTP_COOKIE} "\bJSESSIONID=([^,; ]*)"
RewriteRule . - [E:SESSION=%1]
LogMessage "Honeypot accessed, session \
    %{reqenv:SESSION}"
```

  - The last line should be one long line
  - We extract the JSESSIONID cookie via mod_rewrite
  - mod_rewrite does not rewrite the request but instead sets a request environment variable named SESSION
  - Our log message gets the session id from that variable using "%{reqenv:SESSION}". What's this?

- Excursion: the unified expression parser
  - Traditionally we used expressions in several places having individual and non-consistent implementations:
    - RewriteCond
    - SetEnvIf
    - SSLRequire
    - Allow, Deny
  - We created a universal expression parser which can be used in those places and many more

- Benefits
  - We support by far more types of expressions than in the old implementations
  - Every expression features works in all places
  - The parser is extensible: you can easily add with a few C lines your own functions to it and then use them in existing directives that support the general expression language
  - The parser has a stable API: you can very easily create expression based directives in your own custom modules

- The expression parser has two modes:
  - Returning a boolean value. Expressions are feasible as conditions. That's the most common use.
  - Returning a string. Typically this is used to dynamically interpolate that string in configuration values. That's a more recent feature and not available everywhere. We might also add more ways how to compose strings with the expression parser.
    - Example for string mode: "%{reqenv:MYVAR}" returns the value of the request environment variable "MYVAR"

- Back to custom logging
  - Conditional logging using more complex conditions

```
LogMessage "Access from test network" hook=all \
    "expr=-R '195.226.29.0/25'"
```

    - The last line should be one long line
    - You can add an IP subnet pattern that only matches accesses from your systems and that way activate logging only being executed when you access the system
      - Remember: "hook=all" will log once for every hook so you can follow execution more detailed

- Number 7: “<If EXPRESSION>” – conditional configuration
  - Configuration depending on conditions evaluated at runtime. Typically request based conditions.
  - “<If EXPRESSION>...</If>”, “<Else>...</Else>”, “<ElseIf EXPRESSION>...</ElseIf>

```
<Location /internal>
  <If "-R '195.226.29.0/25'">
    Options +Indexes
  </If>
  <Else>
    Require all denied
  </Else>
</Location>
```

- More on boolean expressions in the configuration
  - Some of the supported terms:
    - *word* in *wordlist*
    - Regular expression */regexp/* oder */regexp/i*
    - String matches *-ipmatch*, *-strmatch*, *-strcmatch*, *-fnmatch*
    - Functions
      - retrieve headers using *req* (Request), *resp* (Response)
      - retrieve table entries using *reqenv*, *osenv*, *notes*
      - convert string case using *tolower*, *toupper*
      - encode strings with percent encoding using *escape*, *unescape*
    - and many more!

- “<If EXPRESSION>” is also nice for debugging
  - You can increase the log level for specific requests

```
<If "-R '195.226.29.0/25'">
  LogLevel trace8
</If>
```

  - Not just IP matches. You can look for a specific user agent, some other header values, session IDs etc.
  - The LogLevel change (and any other configuration inside “<If>”) will only be effective, after the request has been read, so is available for evaluation, but of course before the response gets generated.

- Number 8: “LocationMatch” and other “*Match”: named backreferences
  - “<LocationMatch>”, “<DirectoryMatch>” and “<FilesMatch>” are similar to “<Location>”, “<Directory>” and “<Files>” but one can use regular expressions in the argument (location, directory or file pattern).
  - What if you want to use the matching part in the config block inside “<LocationMatch>” etc., so you want to use $1, $2, …?

- Naming backreferences in “<LocationMatch>”
  - Using $1 etc. will not work, but you can give the matching group real names

```
<LocationMatch "^/(?<context>[^/]+)">
  require ldap-group \
    cn=%{env:MATCH_CONTEXT},ou=contexts,o=Example
</LocationMatch>
```

  - The naming is done with prefixing the group in the match by “?<myname>”. The matching groups will be put into the request environment under the names MATCH_MYNAME, so always “MATCH_” concatenated with the upper case group name.

- Number 9: mod_remoteip: getting the client IP address right
  - If your web server sits behind another reverse proxy, like eg. a load balancer working in reverse proxy mode, your client IP is the IP of the reverse proxy in front of you.
  - How do you get the original client IP address for use in the access log and in access control (“Require ip”)?

- Use mod_remoteip
  - It gets the original client IP from a request header
  - It add it to internal structures, so that in most cases it will be used automatically instead of the connection peer address
    - "Require ip"
    - "%a" in the access log format
      - Caution: per default the log format contains "%h" which does not react to mod_remoteip. Use "%a".

- Which header?
  - Header name is configurable
    - Typically used: “X-Forwarded-For”
      - Example: Apache web server as a reverse proxy automatically adds its client IP address to this header
  - The header can contain a list of IP addresses if there's a chain of proxies in front. mod_remoteip can handle this.

- Security implications
  - Request headers can easily be set (forged) by the client itself
  - So the proxy in front of you should not forward values from outside and instead overwrite the header
  - Therefore it is important to tell mod_remoteip, that it should only use the header as the IP address source if the request comes via a connection from a trusted proxy

- Remoteip minimal example config:

```
# Not needed, default
# RemoteIPHeader X-Forwarded-For
# Adjust to your trusted proxy addresses
RemoteIPInternalProxy 192.168.161.33 ::1
```

- Number 10: mod_ssl: “SSLOpenSSLConfCmd”
  - mod_ssl already has lots of configuration options. More or less one option per OpenSSL feature.
  - The same problem – needing to implement a new configuration option whenever OpenSSL gets a new feature – applies to every software using OpenSSL.
  - The OpenSSL project therefore designed a generic configuration API. Software can send configuration strings to OpenSSL and OpenSSl parses and interpretes them. “SSLOpenSSLConfCmd” uses this new API.

- OpenSSL supported configuration strings
  - For an up-to-date list of supported configuration strings look at the OpenSSL docs, especially at the section “SUPPORTED CONFIGURATION FILE COMMANDS” in the following pages (depending on the OpenSSL version you are using):
    - https://www.openssl.org/docs/man1.0.2/ssl/SSL_CONF_cmd.html
    - https://www.openssl.org/docs/man1.1.0/ssl/SSL_CONF_cmd.html
    - https://www.openssl.org/docs/man1.1.1/man3/SSL_CONF_cmd.html

- Example “SSLOpenSSLConfCmd”

```
SSLOpenSSLConfCmd Options \
  -SessionTicket,ServerPreference
```

  - “-SessionTicket”: switch of support for session tickets (client side TLS session saving and resumption)
    - Alternative: “SSLSessionTickets Off”
  - “ServerPreference”: Use server preferences not client preferences when determining cipher suite, signature algorithm or elliptic curve to use.
    - Alternative: “SSLHonorCipherOrder On”

- More important when using newer OpenSSL versions
  - For 1.0.2 specific directives for most features already exist
  - For 1.1.1 access to new features is immediate without waitingfor new specific mod_ssl directives

- Bonus: look at the docs for “htpasswd” and “rotatelogs”to find interesting new features
  - htpasswd: support for bcrypt password hashing
  - rotatelogs:
    - Time and size based rotation and also the combination of both
    - Optional stable link creation
    - Optional post-rotate processing, eg. compression

Questions?

- Hopefully time for questions …
  - … or send them to rainer.jung@kippdata.de