Appendix A. Rare Nginx Error Messages

We conclude our book with a reference of interesting and not very common error messages that you might encounter in your log files. The table in this appendix may be an emergency reference or another peek into what could go wrong in your setup. In general, Nginx is pretty good at reporting its own problems. The messages usually have a standard format with common items, such as severity, function name, and pointers to external data that caused the problem.

We would recommend against leaving this table unread until a problem occurs because the notes column may contain interesting insights into how Nginx works and help you understand it better. Some of these messages you might not see in your real working experience, which is okay, as the error conditions are exceptions by definition.

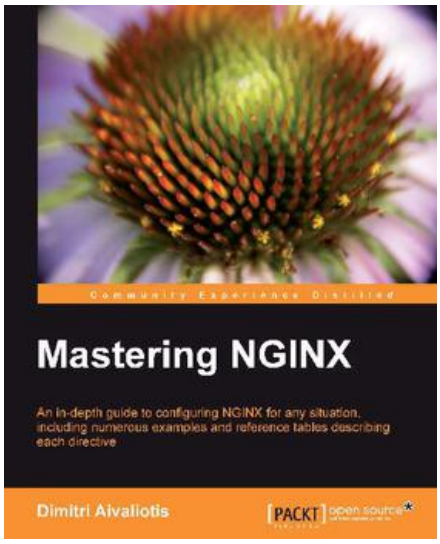
could not open error log file: open() "/var/log/nginx/error.log" failed	This is a very common error, which usually indicates problems with permissions on either the actual log files or the directory structure. You will see this in the `stderr` of the Nginx process because, obviously, it is an error in the error reporting mechanism.
rewrite or internal redirection cycle while internally redirecting	This message means you have a cycle in the rewrite/redirection logic. They may be introduced by complex regular expressions that match too much, for example.
invalid PID number $pid in $file	The saved PID number is garbled. There is a problem with the file that is mentioned in the `pid` directive of your `nginx.conf file` .
getpwnam($user) failed **getgrnam($group) failed**	These two mean that there are problems with the user and group that your Nginx is supposed to run as. This may happen when you try to use configuration files imported from other machines without corrections. See the documentation for the directive at http://nginx.org/en/docs/ngx_core_module.html#user (http://nginx.org/en/docs/ngx_core_module.html#user).
could not build $hash, you should increase $hash_max_size: and **could not build $hash, you should increase $hash_bucket_size:**	These are the messages that Nginx emits when a hash table hits one of two limits—the total hash size and the size of each individual bucket. There are a number of hash tables used throughout the Nginx code, and all of them have the correspondent pairs of directives that look like `*_max_size` and `*_bucket_size` . You have to increase one of those values to get rid of the errors. Also, see the special document about hashes in Nginx at http://nginx.org/en/docs/hash.html (http://nginx.org/en/docs/hash.html).
read()/pread() read only $count of $size from $source	This message means that, unexpectedly, a reading syscall returned less bytes than it should have. There are a number of places where this kind of error may originate.
the configured event method cannot be used with thread pools	Thread pools require the epoll, eventport or the kqueue event subsystem.
pthread_create() failed	This and a number of similar errors come from the thread pool code that uses POSIX threads.
pcre_compile() failed:	Nginx uses the **Perl Compatible Regular Expressions** (**PCRE**) library to implement regexps. PCRE is fine and `pcre_compile()` is the function to compile a regular expression before matching it. Its failure indicates a bad regular expression.

pcre_study() failed: and **JIT compiler does not support pattern:**	Besides simple compilation, PCRE implements several heuristics to optimize the matching of some patterns. That is what `pcre_study()` does. There are very few ways for it to fail, but the JIT compiler, which is one of the optimizations, is a complex piece of software doing much work. Failure inside it probably means either a bug in PCRE or a very weird regular expression.
could not change the accept filter to $value	Accept filters are a feature of BSD kernels that allow postponing the return from the blocking `accept()` calls until there's a meaningful and expected piece of incoming data ready in the buffer. This is an internal error most probably indicating a bug.
$number worker_connections are not enough	You need to increase the number in the directive `worker_connections` .
rename() $filename1 to $filename2 failed before executing new binary process	During the very elaborate process of a graceful executable upgrade, Nginx tried to rename the `pid` file and failed. You may read about how Nginx manages to restart itself without losing connections at http://nginx.org/en/docs/control.html (http://nginx.org/en/docs/control.html). See the USR2 signal.
the number of "worker_processes" is not equal to the number of "worker_cpu_affinity" masks, using last mask for remaining worker processes	CPU affinity is a concept of tying worker processes to particular CPUs. The idea is to be able to say, for example, that the first worker should only run on the first four cores and the second worker should run on the second four cores, respectively. The number of affinity masks that you specify should correspond to the number of worker processes. If it is less, you get this warning message.
no "events" section in configuration	Your configuration file misses one of the most important sections, which is **Events**. See Chapter 1, **Searching for Problems in Nginx Configuration**.
$number worker_connections exceed open file resource limit: $number	The resource limit on the number of open files (file descriptors limit) does not allow having as many worker connections as you wanted by specifying it with the `worker_connections` directive. See the ulimit manpage and also login.conf manpage if you are on FreeBSD.
"ssl_stapling" ignored, issuer certificate not found **"ssl_stapling" ignored, no OCSP responder URL in the certificate** **certificate status not found in the OCSP response** **OCSP responder timed out** **OCSP responder sent invalid "Content-Type" header:**	A number of different messages all mentioning either SSL stapling (and the `ssl_stapling` directive) or OCSP may indicate that your HTTPS works not as efficiently as it could. One of the most complex parts of all X.509 PKI is the issue of certificate revocation. OCSP is the newer attempt at providing online information about the revocation status of certificates, and in the worst case, it requires the client to regularly check the server certificate with an OCSP responder. When OCSP stapling is on, Nginx contacts the responder by itself and provides the clients with a signed, time-stamped OCSP ticket. Basically, a modern HTTPS website should have SSL stapling on and working. Fix these by following all the recommendations in the documentation closely.

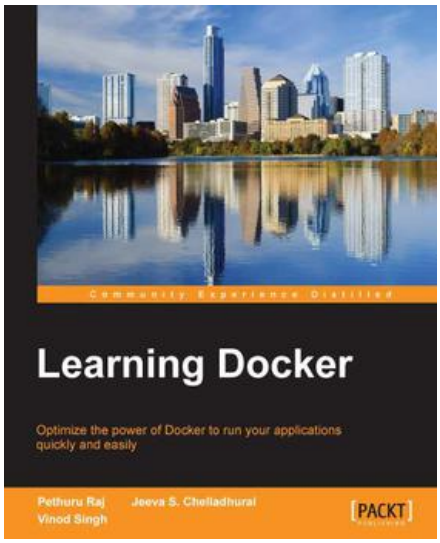
nginx was built with Session Tickets support, however, now it is linked dynamically to an OpenSSL library which has no tlsext support, therefore Session Tickets are not available and also the same about "SNI" instead of Session Tickets	**Server Name Indication** (**SNI**) is an HTTP request `Host:` header counterpart for HTTPS. It is a newer TLS/SSL feature, which permits name-based virtual hosting for HTTPS. The online Nginx documentation has a separate section on SNI at http://nginx.org/en/docs/http/configuring_https_servers.html#sni (http://nginx.org/en/docs/http/configuring_https_servers.html#sni). Session Tickets is a TLS feature-optimizing handshake count. Both of these require OpenSSL support at compile time and at runtime. You may see these error messages when you run Nginx from a binary package on a box with bad OpenSSL.
open(/dev/poll) failed **kqueue() failed** **port_create() failed** **eventfd() failed**	These are all different indications that you have chosen the wrong event subsystem with the directive `use` in the events context. See http://nginx.org/en/docs/events.html (http://nginx.org/en/docs/events.html).
no servers in upstream	Upstreams are groups of backends (whether they are separate hosts or just server software instances running locally) and you specified an empty group.
client intended to send too large body: $number bytes	Well-behaved HTTP clients indicate the size of the requests they send in the `Content-Length:` header. When this size exceeds the value from the `client_max_body_size` directive, Nginx will reject the request with a 413 code. The default value of this limit is only 1 MB, so you may face the problem very often if your website has a function of file uploads.
not well formed XML document	This very vague message is emitted from the rarely used XSLT module. It uses libxml2 and therefore needs valid XML documents.
FastCGI sent in stderr:	This is a message generated by the FastCGI upstream. FastCGI, as a protocol for communications with external processes, provides channels for both `stdout` and `stderr` of the backend software. So this is where `stderr` ends up.
no "proxy_ssl_certificate_key" is defined **no proxy_ssl_trusted_certificate for proxy_ssl_verify**	Modern Nginx has the feature of being a good HTTPS client as well as a server. The HTTP proxy upstream is able to present a client certificate to an HTTPS upstream server. You will need to provide the key to the certificate as well. The client part also can verify the certificate of the server and even check it against a **Certificate Revokation List** (**CRL**).
cache $zone uses the $path cache path while previously it used the $path cache path **cache $zone had previously different levels** **cache file $file is too small**	These messages indicate that the Nginx file cache directory was moved or otherwise tampered with. You should probably clean it and get ready to start again with a cold cache.
duplicate location $location	You have two exactly equal location selectors. Nginx will give you the line number of the second instance, but you will have to find the first yourself.

Next Section (/mapt/book/networking-and-servers/9781785288654/backindex)

Related titles



(/mapt/book/networking_and_servers/9781849517447)



(/mapt/book/virtualization_and_cloud/9781784397937)

(/mapt/book/application_development/9781782168454)