Configuring NGINX workers

NGINX runs a fixed number of worker processes as per the specified configuration. As explained in Chapter 1, **Working with NGINX**, these worker processes are responsible for all request processing. In the following sections, we will work with NGINX worker parameters. These parameters are mostly part of the NGINX global context.

worker_processes

The `worker_processes` directive controls the number of workers:

Copy

```
worker_processes 1;
```

The default value for this is `1`, which means that NGINX runs only one worker. The value should be changed to an optimal value depending on the number of cores available, disks, network subsystem, server load, and so on.

As a starting point, set the value to the number of cores available. Determine the number of cores available using `lscpu`:

Copy

```
$ lscpu
Architecture:     x86_64
CPU op-mode(s):    32-bit, 64-bit
Byte Order:      Little Endian
CPU(s):        4
```

The same can be accomplished by greping out `cpuinfo`:

Copy

```
$ cat /proc/cpuinfo | grep 'processor' | wc -l
```

Now, set this value to the parameter:

Copy

```
# One worker per CPU-core.
worker_processes 4;
```

Alternatively, the directive can have `auto` as its value. This determines the number of cores and spawns an equal number of workers.

When NGINX is running with SSL, it is a good idea to have multiple workers. SSL handshake is blocking in nature and involves disk I/O. Thus, using multiple workers leads to improved performance.

accept_mutex

Since we have configured multiple workers in NGINX, we should also configure the flags that impact worker selection. The `accept_mutex` parameter available under the `events` section will enable each of the available workers to accept new connections one by one. By default, the flag is set to `on`. The following code shows this:

Copy

```
events {
  accept_mutex on;
}
```

If the flag is turned to `off`, all of the available workers will wake up from the waiting state, but only one worker will process the connection. This results in the **Thundering Herd** phenomenon, which is repeated a number of times per second. The phenomenon causes reduced server performance as all the woken-up workers take up CPU time before going back to the wait state. This results in unproductive CPU cycles and nonutilized context switches.

accept_mutex_delay

When `accept_mutex` is enabled, only one worker, which has the mutex lock, accepts connections, while others wait for their turn. The `accept_mutex_delay` corresponds to the timeframe for which the worker would wait, and after which it tries to acquire the mutex lock and starts accepting new connections. The directive is available under the `events` section with a default value of 500 milliseconds. The following code shows this:

Copy

```
events{
  accept_mutex_delay 500ms;
}
```

worker_connections

The next configuration to look at is `worker_connections`, with a default value of `512`. The directive is present under the `events` section. The directive sets the maximum number of simultaneous connections that can be opened by a worker process. The following code shows this:

Copy

```
events{
   worker_connections 512;
}
```

Increase `worker_connections` to something like 1,024 to accept more simultaneous connections.

Note

The value of `worker_connections` does not directly translate into the number of clients that can be served simultaneously. Each browser opens a number of parallel connections to download various components that compose a web page, for example, images, scripts, and so on. Different browsers have different values for this, for example, IE works with two parallel connections while Chrome opens six connections. The number of connections also includes sockets opened with the upstream server, if any.

worker_rlimit_nofile

The number of simultaneous connections is limited by the number of file descriptors available on the system as each socket will open a file descriptor. If NGINX tries to open more sockets than the available file descriptors, it will lead to the `Too many opened files` message in the error.log.

Check the number of file descriptors using `ulimit`:

Copy

```
$ ulimit -n
```

Now, increase this to a value more than `worker_process * worker_connections`. The value should be increased for the user that runs the worker process. Check the user directive to get the username.

NGINX provides the `worker_rlimit_nofile` directive, which can be an alternative way of setting the available file descriptor rather modifying `ulimit`. Setting the directive will have a similar impact to updating `ulimit` for the worker user. The value of this directive overrides the `ulimit` value set for the user. The directive is not present by default. Set a large value to handle large simultaneous connections. The following code shows this:

Copy

```
worker_rlimit_nofile 20960;
```

Tip

To determine the OS limits imposed on a process, read the file `/proc/$pid/limits`. `$pid` corresponds to the PID of the process.

multi_accept

The `multi_accept` flag enables an NGINX worker to accept as many connections as possible when it gets the notification of a new connection. The purpose of this flag is to accept all connections in the listen queue at once. If the directive is disabled, a worker process will accept connections one by one. The following code shows this:

Copy

```
events{
   multi_accept on;
}
```

The directive is available under the `events` section with the default value `off`.

Note

If the server has a constant stream of incoming connections, enabling `multi_accept` may result in a worker accepting more connections than the number specified in `worker_connections`. The overflow will lead to performance loss as the previously accepted connections, part of the overflow, will not get processed.

use

NGINX provides several methods for connection processing. Each of the available methods allows NGINX workers to monitor multiple socket file descriptors, when there is data available for reading/writing. These calls allow NGINX to process multiple socket streams without getting stuck in any one of them. The methods are platform-dependent, and the `configure` command, used to build NGINX, selects the most efficient method available on the platform. If we want to use other methods, they must be enabled first in NGINX.

The `use` directive allows us to override the default method with the method specified. The directive is part of the `events` section:

Copy

```
events {
  use select;
}
```

NGINX supports the following methods of processing connections:

- `select` : This is the standard method of processing connections. It is built automatically on platforms that lack more efficient methods. The module can be enabled or disabled using the `--with-select_module` or `--without-select_module` configuration parameter.
- `poll` : This is the standard method of processing connections. It is built automatically on platforms that lack more efficient methods. The module can be enabled or disabled using the `--with-poll_module` or `--without-poll_module` configuration parameter.
- `kqueue` : This is an efficient method of processing connections available on FreeBSD 4.1, OpenBSD 2.9+, NetBSD 2.0, and OS X.

  There are the additional directives `kqueue_changes` and `kqueue_events` . These directives specify the number of changes and events that NGINX will pass to the kernel. The default value for both of these is `512` .

  Note

  The `kqueue` method will ignore the `multi_accept` directive if it has been enabled.

- `epoll` : This is an efficient method of processing connections available on Linux 2.6+. The method is similar to the FreeBSD `kqueue` .

  There is also the additional directive `epoll_events` . This specifies the number of events that NGINX will pass to the kernel. The default value for this is `512` .

- `/dev/poll` : This is an efficient method of processing connections available on Solaris 7 11/99+, HP/UX 11.22+, IRIX 6.5.15+, and Tru64 UNIX 5.1A+.

  This has the additional directives, `devpoll_events` and `devpoll_changes` . The directives specify the number of changes and events that NGINX will pass to the kernel. The default value for both of these is `32` .

- `eventport` : This is an efficient method of processing connections available on Solaris 10. The method requires necessary security patches to avoid kernel crash issues.
- `rtsig` : Real-time signals is a connection processing method available on Linux 2.2+. The method has some limitations. On older kernels, there is a system-wide limit of 1,024 signals. For high loads, the limit needs to be increased by setting the `rtsig-max` parameter. For kernel 2.6+, instead of the system-wide limit, there is a limit on the number of outstanding signals for each process. NGINX provides the `worker_rlimit_sigpending` parameter to modify the limit for each of the worker processes:

  Copy

  ```
  worker_rlimit_sigpending 512;
  ```

  The parameter is part of the NGINX global configuration.

If the queue overflows, NGINX drains the queue and uses the poll method to process the unhandled events. When the condition is back to normal, NGINX switches back to the `rtsig` method of connection processing. NGINX provides the `rtsig_overflow_events` , `rtsig_overflow_test` , and `rtsig_overflow_threshold` parameters to control how a signal queue is handled on overflows.

The `rtsig_overflow_events` parameter defines the number of events passed to poll.

The `rtsig_overflow_test` parameter defines the number of events handled by poll, after which NGINX will drain the queue.

Before draining the signal queue, NGINX will look up how much it is filled. If the factor is larger than the specified `rtsig_overflow_threshold` , it will drain the queue.

Note

The `rtsig` method requires `accept_mutex` to be set. The method also enables the `multi_accept` parameter.

Next Section (/mapt/book/networking-and-servers/9781785281839/3/ch03lvl1sec19/configuring-nginx-i%2fo)