

Debugging NGINX

Introduction

Debugging helps to identify a bug in the program code if something goes wrong. It is generally used in developing or testing third-party or experimental modules.

NGINX debugging features include the debugging log and creation of a core dump file with its further backtrace.

Configuring NGINX Binary For Debugging

First, you will need to enable debugging in NGINX binary. NGINX Plus already provides you with *nginx-debug* binary while NGINX Open Source requires recompilation.

Configuring F5 NGINX Plus Binary

Starting from [Release 8](#), NGINX Plus ships the *nginx-debug* binary together with the standard binary. To enable debugging in NGINX Plus, you will need to switch from *nginx* to *nginx-debug* binary. Open terminal and run the command:

```
service nginx stop && service nginx-debug start
```

When finished, [enable](#) the debugging log.

Compiling NGINX Open Source

To enable debugging in NGINX Open Source, you will need to recompile the binary with the `--with-debug` option specified in the configure script.

To compile NGINX Open Source with debugging symbols, follow these steps:

1. Download and unpack NGINX source code.
2. Get the list of NGINX configure arguments:

```
nginx -V 2>&1 | grep arguments
```

3. Add the `--with-debug` option to the list of configure commands and run the configure script:

```
./configure --with-debug <other configure arguments>
```

4. Compile and install NGINX:

```
sudo make
sudo make install
```

5. Restart NGINX.

NGINX and Debugging Symbols

Debug symbols helps obtain additional information for debugging, such as functions, variables, data structures, source file and line number information.

NGINX by default is compiled with the “-g” flag that includes debug symbols.

However, if you get the “No symbol table info available” error when you run a [backtrace](#), then debugging symbols are missing and you will need to recompile NGINX with support of debugging symbols.

The exact set of compiler flags depends on the compiler. For example, for the GCC compiler system:

- include debugging symbols with the “-g” flag
- make the debugger output easier to understand by disabling compiler optimization with the “-O0” flag:

```
./configure --with-debug --with-cc-opt='-O0 -g' ...
```

Copy

Enabling Debug Logging in NGINX Configuration

The debugging log records errors and any debugging-related information and is disabled by default. To enable it, make sure NGINX is compiled to support debugging (see [Configuring NGINX Binary For Debugging](#)) and then enable it in NGINX configuration file with the `debug` parameter of the `error_log` directive. The debugging log may be written to a file, an allocated *buffer* in memory, *stderr* output, or to *syslog*.

It is recommended enabling the debugging log on the “*main*” level of NGINX configuration to get the full picture of what’s going on.

Writing the Debugging Log to a File

Writing the debugging log to a file may slow down performance under high load. Also note that the file can grow very large and quickly eat up disk space. To reduce the negative impact, you can configure the debugging log to be written into a memory buffer, or set the debugging log for particular IP addresses. See [Writing the Debugging Log to Memory](#) and [Debug Log for Selected IPs](#) for details.

To enable writing the debugging log to a file:

1. Make sure your NGINX is configured with the `--with-debug` configuration option. Run the command and check if the output contains the `--with-debug` line:

```
nginx -V 2>&1 | grep -- '--with-debug'
```

Copy

2. Open NGINX configuration file:

```
sudo vi /etc/nginx/nginx.conf
```

Copy

3. Find the `error_log` directive which is used to write the log. Enable the `debug` parameter and specify the path to the log file:

```
error_log /var/log/nginx/error.log debug;
```

Copy

4. Save the configuration and exit the configuration file.

We value your privacy

To provide the best experience, we leverage third-party technologies to personalize what you see, and to better understand what content is important to you. View our [privacy policy](#) for details.

Writing the Debugging Log to Memory

The debugging log can be written to a memory using a cyclic buffer. The advantage is that logging on the debug level will not have significant impact on performance under high load.

To enable writing the debug log to memory:

1. Make sure your NGINX is configured with the `--with-debug` configuration option. Run the command and check if the output contains the `--with-debug` line:

```
nginx -V 2>&1 | grep -- '--with-debug'
```

Copy

2. In NGINX configuration file, enable a memory buffer for debug logging with the `error_log` directive specified in the `main` context:

```
error_log memory:32m debug;
...
http {
    ...
}
```

Copy

Extracting Debug Log From Memory

The log can be extracted from the memory buffer using a script executed in the GDB debugger.

To extract the debugging log from memory:

1. Obtain the PID of NGINX worker process:

```
ps axu |grep nginx
```

2. Launch the GDB debugger:

```
sudo gdb -p <nginx PID obtained at the previous step>
```

3. Copy the script, paste it to GDB and press “Enter”. The script will save the log in the *debug_log.txt* file located in the current directory:

```
set $log = ngx_cycle->log
while $log->writer != ngx_log_memory_writer
    set $log = $log->next
end
set $buf = (ngx_log_memory_buf_t *) $log->wdata
dump binary memory debug_log.txt $buf->start $buf->end
```

4. Quit GDB by pressing CTRL+D.

5. Open the file “*debug_log.txt*” located in the current directory:


```
sudo less debug_log.txt
```

Debug Log for Selected IP

It is possible to enable the debugging log for a particular IP in a production environment as it will not negatively affect the performance. The `debug_connection` directive can be defined more than once.

We value your privacy

To provide the best experience, we leverage third-party technologies to personalize what you see, and to better understand what content is important to you. View our [privacy policy](#) for details.



```
error_log /path/to/log;
...
events {
    debug_connection 192.168.1.1;
    debug_connection 192.168.10.0/24;
}
```

Debug Log for Each Virtual Host

Generally, the `error_log` directive is specified in the `main` context and thus is applied to all other contexts including `server` and `location`. But if there is another `error_log` directive specified inside a particular `server` or a `location` block, the global settings will be overridden and such `error_log` directive will set its own path to the log file and the level of logging.

To set up the debugging log for a particular virtual host, add the `error_log` directive inside a particular `server` block, in which set a new path to the log file and the `debug` logging level:

```
error_log /path1/to/log debug;
...
http {
    ...
    server {
        error_log /path2/to/log debug;
        ...
    }
}
```

To disable the debugging log per a particular virtual host, specify the `error_log` directive inside a particular `server` block, and specify a path to the log file only:

Copy

```
error_log /path/to/log debug;

...

http {
    ...
    server {
        error_log /path/to/log;
        ...
    }
}
```

Enabling Core Dumps

A core dump file can help identify and fix problems that are causing NGINX to crash. A core dump file may contain sensitive information such as passwords and private keys, so ensure that they are treated securely.

In order to create a core dump file, they must be enabled in both the operating system and the NGINX configuration file.

Enabling Core Dumps in the Operating System

Perform the following steps in your operating system:

1. Specify a working directory in which a core dump file will be saved, for example, *"/tmp/cores"*:

Copy

```
mkdir /tmp/cores
```

2. Make sure the directory is writable by NGINX worker process:

We value your privacy

To provide the best experience, we leverage third-party technologies to personalize what you see, and to better understand what content is important to you. View our [privacy policy](#) for details.

Copy

```
sudo chown root:root /tmp/cores
sudo chmod 1777 /tmp/cores
```

3. Disable the limit for the maximum core file size:

Copy

```
sudo prlimit --core=unlimited
```

If the operation ends up with “C

Copy

```
sudo sh -c "ulimit -c unlimited && exec su $LOGNAME"
```

4. Enable core dumps for the *setuid* and *setgid* processes.

For CentOS 7.0, Debian 8.2, Ubuntu 14.04, run the commands:

Copy

```
echo "/tmp/cores/core.%e.%p" | sudo tee /proc/sys/kernel/core_pattern
sudo sysctl -w fs.suid_dumpable=2
sysctl -p
```

For FreeBSD, run the commands:

Copy

```
sudo sysctl kern.sugid_coredump=1
sudo sysctl kern.corefile=/tmp/cores/%N.core.%P
```

Enabling Core Dumps in NGINX Configuration

To enable core dumps in the NGINX configuration file:

1. Open the NGINX configuration file:

Copy

```
sudo vi /usr/local/etc/nginx/nginx.conf
```

2. Define a directory that will keep core dump files with the `working_directory` directive. The directive is specified on the *main* configuration level:

```
working_directory /tmp/cores/;
```

3. Make sure the directory exists and is writable by NGINX worker process. Open terminal and run the commands:

```
sudo chown root:root /tmp/cores
sudo chmod 1777 /tmp/cores
```

4. Specify the maximum possible size of the core dump file with the `worker_rlimit_core` directive. The directive is also specified on the `main` configuration level. If the core dump file size exceeds the value, the core dump file will not be created.

```
worker_rlimit_core 500M;
```

Example:

```
worker_processes auto;
error_log /var/log/nginx/error.log debug;
working_directory /tmp/cores/;
worker_rlimit_core 500M;

events {
    ...
}

http {
    ...
}
```

We value your privacy

To provide the best experience, we leverage third-party technologies to personalize what you see, and to better understand what content is important to you. View our [privacy policy](#) for details.

With these settings, a core dump file will not exceed 500 megabytes.

Obtaining Backtraces

Backtraces provide information from a core dump file about what was wrong when a program crashed.

To get a backtrace from a core dump file:

1. Open a core dump file with the GDB debugger using the pattern:

```
sudo gdb <nginx_executable_path> <coredump_file_path>
```

2. Type-in the “*backtrace*” command to get a stack trace from the time of the crash:

```
(gdb) backtrace
```

If the “*backtrace*” command resulted with the “No symbol table info available” message, you will need to recompile NGINX binary to include debugging symbols. See [NGINX and Debugging Symbols](#).

Dumping NGINX Configuration From a Running Process

You can extract the current NGINX configuration from the master process in memory. This can be useful when you need to:

- verify which configuration has been loaded
- restore a previous configuration if the version on disk has been accidentally removed or overwritten

The configuration dump can be obtained with a GDB script provided that your NGINX has the debug support.

1. Make sure your NGINX is built with the debug support (the `--with-debug` configure option in the list of the configure arguments). Run the command and check if the output contains the `--with-debug` line:

```
nginx -V 2>&1 | grep -- '--with-debug'
```

2. Obtain the PID of NGINX worker process:

```
ps axu | grep nginx
```

3. Launch the GDB debugger:

```
sudo gdb -p <nginx PID obtained at the previous step>
```

4. Copy and paste the script to GDB and press “Enter”. The script will save the configuration in the *nginx_conf.txt* file in the current directory:

```
set $cd = ngx_cycle->config_dump
set $nelts = $cd.nelts
set $elts = (ngx_conf_dump_t*) ($cd.elts)
while ($nelts-- > 0)
set $name = $elts[$nelts]->name.data
printf "Dumping %s to nginx_conf.txt\n", $name
append memory nginx_conf.txt \
    $elts[$nelts]->buffer.start $elts[$nelts]->buffer.end
end
```

5. Quit GDB by pressing *CTRL+D*.

6. Open the file *nginx_conf.txt* located in the current directory:

```
sudo vi nginx_conf.txt
```

Asking for help

When asking for help with debugging, please provide the following information:

1. NGINX version, compiler version

```
nginx -V
```

2. Current full NGINX configuration. See [Dumping NGINX Configuration From a Running Process](#)

3. The debugging log. See [Enabling Debug Logging in NGINX Configuration](#)

4. The obtained backtrace. See [Enabling Core Dumps, Obtaining Backtrace](#)

Last modified September 24, 2024



Found a bug? Looking for something new?

LET US KNOW

Company

About F5 NGINX

Products

F5 NGINX One

NGINX on GitHub

NGINX Open Source

Social



Events	F5 NGINX Plus	NGINX Unit	
	F5 NGINX App Protect	NGINX Amplify	
Resources	F5 NGINX Instance Manager	NGINX Agent	
Blog	F5 NGINX Ingress Controller	NGINX Kubernetes Ingress Controller	
FAQ	F5 NGINX Gateway Fabric	NGINX Gateway Fabric	
Professional Services	F5 NGINXaaS for Azure		
Training			

We value your privacy

To provide the best experience, we leverage third-party technologies to personalize what you see, and to better understand what content is important to you. View our [privacy policy](#) for details.