Installing NGINX Open Source

Install NGINX Open Source either as a prebuilt package or from source, following step-by-step instructions for all supported Linux distributions.

This article explains how to install NGINX Open Source.

Choosing Between a Stable or a Mainline Version

NGINX Open Source is available in two versions:

- **Mainline** – Includes the latest features and bug fixes and is always up to date. It is reliable, but it may include some experimental modules, and it may also have some number of new bugs.
- **Stable** – Doesn't include all of the latest features, but has critical bug fixes that are always backported to the mainline version. We recommend the stable version for production servers.

Choosing Between a Prebuilt Package and Compiling from Source

Both the NGINX Open Source mainline and stable versions can be installed in two ways:

- As a prebuilt binary package. This is a quick and easy way to install NGINX Open Source. The package includes almost all official NGINX modules and is available for most popular operating systems. See Installing a Prebuilt Package.
- As binaries you compile from source. This way is more flexible: you can add particular modules, including third-party modules, or apply the latest security patches. See Compiling and Installing from Source for details.

Installing a Prebuilt Package

Installing NGINX Open Source from a package is much easier and faster than building from source, but building from source enables you to compile in non-standard modules. Prebuilt packages are available for most popular Linux distributions, including CentOS, Debian, Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), and Ubuntu. See Linux packages at **nginx.org** for the list of currently supported operating systems.

Modules Included in a Prebuilt Package

See Source packages at **nginx.org** for the list of modules included in each prebuilt package.

Installing Prebuilt RHEL, CentOS, Oracle Linux, AlmaLinux, Rocky Linux Packages

NGINX, Inc. provides packages for the following CentOS, Oracle Linux, RHEL, AlmaLinux and Rocky Linux versions:

Version	Supported Platforms
7.4+	x86_64, aarch64/arm64
8x	x86_64, aarch64/arm64, s390x
9x	x86_64, aarch64/arm64, s390x

The package can be installed from:

- A default RHEL / CentOS / Oracle Linux / AlmaLinux / Rocky Linux repository. This is the quickest way, but generally the provided package is outdated.
- The official repo at **nginx.org**. You have to set up the `yum` repository the first time, but after that the provided package is always up to date.

Installing a Prebuilt CentOS/RHEL/ Oracle Linux/AlmaLinux/Rocky Linux Package from an OS Repository

1. Install the EPEL repository:

   ```
   sudo yum install epel-release
   ```

2. Update the repository:

   ```
   sudo yum update
   ```

3. Install NGINX Open Source:

   ```
   sudo yum install nginx
   ```

4. Verify the installation:

   ```
   sudo nginx -v
   ```

Installing a Prebuilt RHEL/CentOS/Oracle Linux/AlmaLinux/Rocky Linux Package from the Official NGINX Repository

1. Install the prerequisites:

```
sudo yum install yum-utils
```

2. Set up the `yum` repository for RHEL/CentOS/Oracle Linux/AlmaLinux/Rocky Linux by creating the file **nginx.repo** in **/etc/yum.repos.d**, for example using `vi`:

   ```
   sudo vi /etc/yum.repos.d/nginx.repo
   ```

3. Add the following lines to **nginx.repo**:

   ```
   [nginx-stable]
   name=nginx stable repo
   baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
   gpgcheck=1
   enabled=1
   gpgkey=https://nginx.org/keys/nginx_signing.key
   module_hotfixes=true

   [nginx-mainline]
   name=nginx mainline repo
   baseurl=http://nginx.org/packages/mainline/centos/$releasever/$basearch/
   gpgcheck=1
   enabled=0
   gpgkey=https://nginx.org/keys/nginx_signing.key
   module_hotfixes=true
   ```

   where he `stable` or `mainline` element points to the latest stable or mainline version of NGINX Open Source. By default, the repository for stable nginx packages is used. If you would like to use mainline nginx packages, run the following command:

   ```
   sudo yum-config-manager --enable nginx-mainline
   ```

4. Save the changes and quit `vi` (press ESC and type `wq` at the `:` prompt).

5. Update the repository:

   ```
   sudo yum update
   ```

6. Install the NGINX Open Source package:

   ```
   sudo yum install nginx
   ```

   When prompted to accept the GPG key, verify that the fingerprint matches `573B FD6B 3D8F BC64 1079 A6AB ABF5 BD82 7BD9 BF62`, and if so, accept it.

7. Start NGINX Open Source:

   ```
   sudo nginx
   ```

8. Verify that NGINX Open Source is up and running:

   ```
   curl -I 127.0.0.1
   HTTP/1.1 200 OK
   Server: nginx/1.23.4
   ```

Installing Prebuilt Debian Packages

NGINX provides packages for the following Debian operating systems:

Version	Codename	Supported Platforms
11.x	bullseye	x86_64, aarch64/arm64

The package can be installed from:

- A default Debian repository. This is the quickest way, but generally the provided package is outdated.
- The official repo at **nginx.org**. You have to set up the `apt-get` repository the first time, but after that the provided package is always up to date.

Installing a Prebuilt Debian Package from an OS Repository

1. Update the Debian repository information:

```
sudo apt-get update
```

2. Install the NGINX Open Source package:

```
sudo apt-get install nginx
```

3. Verify the installation:

```
sudo nginx -v
```

Installing a Prebuilt Debian Package from the Official NGINX Repository

1. Install the prerequisites:

```
sudo apt install curl gnupg2 ca-certificates lsb-release debian-archive-keyring
```

2. Import an official nginx signing key so `apt` could verify the packages authenticity. Fetch the key:

```
curl https://nginx.org/keys/nginx_signing.key | gpg --dearmor \
    | sudo tee /usr/share/keyrings/nginx-archive-keyring.gpg >/dev/null
```

3. Verify that the downloaded file contains the proper key:

```
gpg --dry-run --quiet --no-keyring --import --import-options import-show /usr/share/keyrings/nginx-archive-keyring.gpg
```

The output should contain the full fingerprint `573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62` as follows:

```
pub   rsa2048 2011-08-19 [SC] [expires: 2024-06-14]
      573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62
uid                      nginx signing key <signing-key@nginx.com>
```

If the fingerprint is different, remove the file.

4. To set up the `apt` repository for stable nginx packages, run the following command:

```
echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg] \
http://nginx.org/packages/debian `lsb_release -cs` nginx" \
    | sudo tee /etc/apt/sources.list.d/nginx.list
```

If you would like to use `mainline` nginx packages, run the following command instead:

```
echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg] \
http://nginx.org/packages/mainline/debian `lsb_release -cs` nginx" \
    | sudo tee /etc/apt/sources.list.d/nginx.list
```

5. Set up repository pinning to prefer our packages over distribution-provided ones:

```
echo -e "Package: *\nPin: origin nginx.org\nPin: release o=nginx\nPin-Priority: 900\n" \
    | sudo tee /etc/apt/preferences.d/99nginx
```

6. Install the NGINX package:

```
sudo apt update
sudo apt install nginx
```

7. Start NGINX Open Source:

```
sudo nginx
```

8. Verify that NGINX Open Source is up and running:

```
curl -I 127.0.0.1
HTTP/1.1 200 OK
Server: nginx/1.23.4
```

Installing Prebuilt Ubuntu Packages

NGINX provides packages for the following Ubuntu operating systems:

Version	Codename	Supported Platforms
18.04	bionic	x86_64, aarch64/arm64
20.04	focal	x86_64, aarch64/arm64, s390x
22.04	jammy	x86_64, aarch64/arm64, s390x
22.10	kinetic	x86_64, aarch64/arm64

The package can be installed from:

- A default Ubuntu repository. This is the quickest way, but generally the provided package is outdated.
- The official repo at **nginx.org**. You have to set up the `apt-get` repository the first time, but after that the provided package is always up to date.

Installing a Prebuilt Ubuntu Package from an Ubuntu Repository

1. Update the Ubuntu repository information:

   ```
   sudo apt-get update
   ```

2. Install the package:

   ```
   sudo apt-get install nginx
   ```

3. Verify the installation:

   ```
   sudo nginx -v
   ```

Installing a Prebuilt Ubuntu Package from the Official NGINX Repository

1. Install the prerequisites:

   ```
   sudo apt install curl gnupg2 ca-certificates lsb-release ubuntu-keyring
   ```

2. Import an official nginx signing key so apt could verify the packages authenticity. Fetch the key:

   ```
   curl https://nginx.org/keys/nginx_signing.key | gpg --dearmor \
   | sudo tee /usr/share/keyrings/nginx-archive-keyring.gpg >/dev/null
   ```

3. Verify that the downloaded file contains the proper key:

   ```
   gpg --dry-run --quiet --no-keyring --import --import-options import-show /usr/share/keyrings/nginx-archive-keyring.gpg
   ```

   The output should contain the full fingerprint `573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62` as follows:

   ```
   pub   rsa2048 2011-08-19 [SC] [expires: 2024-06-14]
     573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62
   uid                      nginx signing key <signing-key@nginx.com>
   ```

   If the fingerprint is different, remove the file.

4. To set up the `apt` repository for stable nginx packages, run the following command:

   ```
   echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg] \
   http://nginx.org/packages/ubuntu `lsb_release -cs` nginx" \
       | sudo tee /etc/apt/sources.list.d/nginx.list
   ```

   If you would like to use `mainline` nginx packages, run the following command instead:

   ```
   echo "deb [signed-by=/usr/share/keyrings/nginx-archive-keyring.gpg] \
   http://nginx.org/packages/mainline/ubuntu `lsb_release -cs` nginx" \
       | sudo tee /etc/apt/sources.list.d/nginx.list
   ```

5. Set up repository pinning to prefer our packages over distribution-provided ones:

   ```
   echo -e "Package: *\nPin: origin nginx.org\nPin: release o=nginx\nPin-Priority: 900\n" \
       | sudo tee /etc/apt/preferences.d/99nginx
   ```

6. Install NGINX Open Source:

```
sudo apt update
sudo apt install nginx
```

7. Start NGINX Open Source:

```
sudo nginx
```

8. Verify that NGINX Open Source is up and running:

```
curl -I 127.0.0.1
HTTP/1.1 200 OK
Server: nginx/1.23.4
```

Installing SUSE Packages

NGINX provides packages for SUSE Linux Enterprise Server:

Version	Supported Platforms
SLES 12 SP5+	x86_64
SLES 15 SP2+	x86_64

Installing a Prebuilt SUSE Package from the Official NGINX Repository

1. Install the prerequisites:

```
sudo zypper install curl ca-certificates gpg2
```

2. To set up the `zypper` repository for stable nginx packages, run the following command:

```
sudo zypper addrepo --gpgcheck --type yum --refresh --check \
    'http://nginx.org/packages/sles/$releasever_major' nginx-stable
```

3. If you would like to use mainline nginx packages, run the following command instead:

```
sudo zypper addrepo --gpgcheck --type yum --refresh --check \
    'http://nginx.org/packages/mainline/sles/$releasever_major' nginx-mainline
```

4. Import an official nginx signing key so `zypper/rpm` could verify the packages authenticity. Fetch the key:

```
curl -o /tmp/nginx_signing.key https://nginx.org/keys/nginx_signing.key
```

5. Verify that the downloaded file contains the proper key:

```
gpg --with-fingerprint /tmp/nginx_signing.key
```

The output should contain the full fingerprint `573B FD6B 3D8F BC64 1079 A6AB ABF5 BD82 7BD9 BF62` as follows:

```
pub  2048R/7BD9BF62 2011-08-19 [expires: 2024-06-14]
      Key fingerprint = 573B FD6B 3D8F BC64 1079  A6AB ABF5 BD82 7BD9 BF62
uid nginx signing key <signing-key@nginx.com>
```

6. Import the key to the rpm database:

```
sudo rpmkeys --import /tmp/nginx_signing.key
```

7. To install nginx, run the command:

```
sudo zypper install nginx
```

Installing Prebuilt Alpine Linux Packages

NGINX provides packages for the following Alpine Linux operating systems:

Version	Supported Platforms
3.14	x86_64, aarch64/arm64

Version	Supported Platforms
3.15	x86_64, aarch64/arm64
3.16	x86_64, aarch64/arm64
3.17	x86_64, aarch64/arm64

The package can be installed from the official repo at **nginx.org**. You have to set up the `apt-get` repository the first time, but after that the provided package is always up to date.

Installing a Prebuilt Alpine Linux Package from the Official NGINX Repository

1. Install the prerequisites:

   ```
   sudo apk add openssl curl ca-certificates
   ```

2. To set up the apk repository for stable nginx packages, run the command:

   ```
   printf "%s%s%s\n" \
   "http://nginx.org/packages/alpine/v" \
   `egrep -o '^[0-9]+\.[0-9]+' /etc/alpine-release` \
   "/main" \
   | sudo tee -a /etc/apk/repositories
   ```

   For mainline nginx packages, run the following command instead:

   ```
   printf "%s%s%s\n" \
   "http://nginx.org/packages/mainline/alpine/v" \
   `egrep -o '^[0-9]+\.[0-9]+' /etc/alpine-release` \
   "/main" \
   | sudo tee -a /etc/apk/repositories
   ```

3. Import an official nginx signing key so apk could verify the packages authenticity. Fetch the key:

   ```
   curl -o /tmp/nginx_signing.rsa.pub https://nginx.org/keys/nginx_signing.rsa.pub
   ```

4. Verify that the downloaded file contains the proper key:

   ```
   openssl rsa -pubin -in /tmp/nginx_signing.rsa.pub -text -noout
   ```

   The output should contain the following modulus:

   ```
   Public-Key: (2048 bit)
   Modulus:
       00:fe:14:f6:0a:1a:b8:86:19:fe:cd:ab:02:9f:58:
       2f:37:70:15:74:d6:06:9b:81:55:90:99:96:cc:70:
       5c:de:5b:e8:4c:b2:0c:47:5b:a8:a2:98:3d:11:b1:
       f6:7d:a0:46:df:24:23:c6:d0:24:52:67:ba:69:ab:
       9a:4a:6a:66:2c:db:e1:09:f1:0d:b2:b0:e1:47:1f:
       0a:46:ac:0d:82:f3:3c:8d:02:ce:08:43:19:d9:64:
       86:c4:4e:07:12:c0:5b:43:ba:7d:17:8a:a3:f0:3d:
       98:32:b9:75:66:f4:f0:1b:2d:94:5b:7c:1c:e6:f3:
       04:7f:dd:25:b2:82:a6:41:04:b7:50:93:94:c4:7c:
       34:7e:12:7c:bf:33:54:55:47:8c:42:94:40:8e:34:
       5f:54:04:1d:9e:8c:57:48:d4:b0:f8:e4:03:db:3f:
       68:6c:37:fa:62:14:1c:94:d6:de:f2:2b:68:29:17:
       24:6d:f7:b5:b3:18:79:fd:31:5e:7f:4c:be:c0:99:
       13:cc:e2:97:2b:dc:96:9c:9a:d0:a7:c5:77:82:67:
       c9:cb:a9:e7:68:4a:e1:c5:ba:1c:32:0e:79:40:6e:
       ef:08:d7:a3:b9:5d:1a:df:ce:1a:c7:44:91:4c:d4:
       99:c8:88:69:b3:66:2e:b3:06:f1:f4:22:d7:f2:5f:
       ab:6d
   Exponent: 65537 (0x10001)
   ```

5. Move the key to `apk` trusted keys storage:

   ```
   sudo mv /tmp/nginx_signing.rsa.pub /etc/apk/keys/
   ```

6. To install nginx, run the command:

```
sudo apk add nginx
```

The `@nginx` tag should also be specified when installing packages with dynamic modules:

```
sudo apk add nginx-module-image-filter@nginx nginx-module-njs@nginx
```

Installing Prebuilt Amazon Linux Packages

NGINX provides packages for

- Amazon Linux 2 (LTS) x86_64, aarch64/arm64
- Amazon Linux 2023 x86_64, aarch64/arm64

Installing a Prebuilt Alpine Linux Package from the Official NGINX Repository

1. Install the prerequisites:

```
sudo yum install yum-utils
```

2. To set up the `yum` repository for Amazon Linux 2, create the file named `/etc/yum.repos.d/nginx.repo` with the following contents:

```
[nginx-stable]
name=nginx stable repo
baseurl=http://nginx.org/packages/amzn2/$releasever/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true

[nginx-mainline]
name=nginx mainline repo
baseurl=http://nginx.org/packages/mainline/amzn2/$releasever/$basearch/
gpgcheck=1
enabled=0
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true
```

To set up the `yum` repository for Amazon Linux 2023, create the file named `/etc/yum.repos.d/nginx.repo` with the following contents:

```
[nginx-stable]
name=nginx stable repo
baseurl=http://nginx.org/packages/amzn/2023/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true

[nginx-mainline]
name=nginx mainline repo
baseurl=http://nginx.org/packages/mainline/amzn/2023/$basearch/
gpgcheck=1
enabled=0
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true
```

By default, the repository for `stable` nginx packages is used. If you would like to use `mainline` nginx packages, run the following command:

```
sudo yum-config-manager --enable nginx-mainline
```

3. Install nginx:

```
sudo yum install nginx
```

When prompted to accept the GPG key, verify that the fingerprint matches `573B FD6B 3D8F BC64 1079 A6AB ABF5 BD82 7BD9 BF62`, and if so, accept it.

Compiling and Installing from Source

Compiling NGINX Open Source from source affords more flexibility than prebuilt packages: you can add particular modules (from NGINX or third parties), and apply latest security patches.

Installing NGINX Dependencies

Prior to compiling NGINX Open Source from source, you need to install libraries for its dependencies:

- PCRE – Supports regular expressions. Required by the NGINX Core and Rewrite modules.

  ```
  wget github.com/PCRE2Project/pcre2/releases/download/pcre2-10.40/pcre2-10.40.tar.gz
  tar -zxf pcre2-10.40.tar.gz
  cd pcre2-10.40
  ./configure
  make
  sudo make install
  ```

- zlib – Supports header compression. Required by the NGINX Gzip module.

  ```
  wget http://zlib.net/zlib-1.2.13.tar.gz
  tar -zxf zlib-1.2.13.tar.gz
  cd zlib-1.2.13
  ./configure
  make
  sudo make install
  ```

- OpenSSL – Supports the HTTPS protocol. Required by the NGINX SSL module and others.

  ```
  wget http://www.openssl.org/source/openssl-1.1.1t.tar.gz
  tar -zxf openssl-1.1.1t.tar.gz
  cd openssl-1.1.1t
  ./Configure darwin64-x86_64-cc --prefix=/usr
  make
  sudo make install
  ```

Downloading the Sources

Download the source files for both the stable and mainline versions from **nginx.org**.

To download and unpack the source for the latest *mainline* version, run:

```
wget https://nginx.org/download/nginx-1.23.4.tar.gz
tar zxf nginx-1.23.4.tar.gz
cd nginx-1.23.4
```

To download and unpack source files for the latest *stable* version, run:

```
wget https://nginx.org/download/nginx-1.24.0.tar.gz
tar zxf nginx-1.24.0.tar.gz
cd nginx-1.24.0
```

Configuring the Build Options

Configure options are specified with the `./configure` script that sets up various NGINX parameters, including paths to source and configuration files, compiler options, connection processing methods, and the list of modules. The script finishes by creating the `Makefile` required to compile the code and install NGINX Open Source.

An example of options to the `configure` script (should be typed as a single line):

```
./configure
--sbin-path=/usr/local/nginx/nginx
--conf-path=/usr/local/nginx/nginx.conf
--pid-path=/usr/local/nginx/nginx.pid
--with-pcre=../pcre2-10.40
--with-zlib=../zlib-1.2.13
--with-http_ssl_module
--with-stream
--with-mail=dynamic
--add-module=/usr/build/nginx-rtmp-module
--add-dynamic-module=/usr/build/3party_module
```

Configuring NGINX Paths

The `configure` script allows you to set paths to NGINX binary and configuration files, and to dependent libraries such as PCRE or SSL, in order to link them statically to the NGINX binary.

Parameter	Description
`--prefix=<PATH>`	Directory for NGINX files, and the base location for all relative paths set by the other `configure` script options (excluding paths to libraries) and for the path to the **nginx.conf** configuration file. Default: **/usr/local/nginx**.
`--sbin-path=<PATH>`	Name of the NGINX executable file, which is used only during installation. Default: **/sbin/nginx
`--conf-path=<PATH>`	Name of the NGINX configuration file. You can, however, always override this value at startup by specifying a different file with the `-c <FILENAME>` option on the `nginx` command line. Default: **conf/nginx.conf
`--pid-path=<PATH>`	Name of the **nginx.pid** file, which stores the process ID of the `nginx` master process. After installation, the path to the filename can be changed with the pid directive in the NGINX configuration file. Default: **/logs/nginx.pid
`--error-log-path=<PATH>`	Name of the primary log file for errors, warnings, and diagnostic data. After installation, the filename can be changed with the error_log directive in the NGINX configuration file. Default: **/logs/error.log
`--http-log-path=<PATH>`	Name of the primary log file for requests to the HTTP server. After installation, the filename can always be changed with the access_log directive in the NGINX configuration file. Default: **/logs/access.log
`--user=<NAME>`	Name of the unprivileged user whose credentials are used by the NGINX worker processes. After installation, the name can be changed with the user directive in the NGINX configuration file. Default: `nobody`
`--group=<NAME>`	Name of the group whose credentials are used by the NGINX worker processes. After installation, the name can be changed with the user directive in the NGINX configuration file. Default: the value set by the `--user` option.
`--with-pcre=<PATH>`	Path to the source for the PCRE library, which is required for regular expressions support in the location directive and the Rewrite module.
`--with-pcre-jit`	Builds the PCRE library with "just-in-time compilation" support (the pcre_jit directive).
`--with-zlib=<PATH>`	Path to the source for the `zlib` library, which is required by the Gzip module.

Configuring NGINX GCC Options

With the `configure` script you can also specify compiler-related options.

Parameter	Description
`--with-cc-opt="<PARAMETERS>"`	Additional parameters that are added to the `CFLAGS` variable. When using the system PCRE library under FreeBSD, the mandatory value is `--with-cc-opt="-I /usr/local/include"`. If the number of files supported by `select()` needs to be increased, it can also specified here as in this example: `--with-cc-opt="-D FD_SETSIZE=2048"`.
`--with-ld-opt="<PARAMETERS>"`	Additional parameters that are used during linking. When using the system PCRE library under FreeBSD, the mandatory value is `--with-ld-opt="-L /usr/local/lib"`.

Specifying NGINX Connection Processing Methods

With the `configure` script you can redefine the method for event-based polling. For more information, see Connection processing methods in the NGINX reference documentation.

Module Name	Description
`--with-select_module`, `--without-select_module`	Enables or disables building a module that enable NGINX to work with the `select()` method. The modules is built automatically if the platform does not appear to support more appropriate methods such as `kqueue`, `epoll`, or `/dev/poll`.
`--with-poll_module`, `--without-poll_module`	Enables or disables building a module that enables NGINX to work with the `poll()` method. The module is built automatically if the platform does not appear to support more appropriate methods such as `kqueue`, `epoll`, or `/dev/poll`.

Selecting the NGINX Modules to Build

NGINX consists of a set of function-specific *modules*, which are specified with `configure` script along with other build options.

Some modules are built by default – they do not have to be specified with the `configure` script. Default modules can however be explicitly excluded from the NGINX binary with the `--without-<MODULE-NAME>` option on the `configure` script.

Modules not included by default, as well as third-party modules, must be explicitly specified in the `configure` script together with other build options. Such modules can be linked to NGINX binary either *statically* (they are then loaded each time NGINX starts) or *dynamically* (they are loaded only if associated directives are included in the NGINX configuration file.

Modules Built by Default

If you do not need a module that is built by default, you can disable it by naming it with the `--without-<MODULE-NAME>` option on the `configure` script, as in this example which disables the Empty GIF module (should be typed as a single line):

```
./configure
--sbin-path=/usr/local/nginx/nginx
--conf-path=/usr/local/nginx/nginx.conf
--pid-path=/usr/local/nginx/nginx.pid
--with-http_ssl_module
--with-stream
--with-pcre=../pcre2-10.40
--with-zlib=../zlib-1.2.13
--without-http_empty_gif_module
```

Module Name	Description
http_access_module	Accepts or denies requests from specified client addresses.
http_auth_basic_module	Limits access to resources by validating the user name and password using the HTTP Basic Authentication protocol.
http_autoindex_module	Processes requests ending with the forward-slash character (/) and produces a directory listing.
http_browser_module	Creates variables whose values depend on the value of the `User-Agent` request header.
http_charset_module	Adds the specified character set to the `Content-Type` response header. Can convert data from one character set to another.
http_empty_gif_module	Emits a single-pixel transparent GIF.
http_fastcgi_module	Passes requests to a FastCGI server.
http_geo_module	Creates variables with values that depend on the client IP address.
http_gzip_module	Compresses responses using `gzip`, reducing the amount of transmitted data by half or more.
http_limit_conn_module	Limits the number of connections per a defined key, in particular, the number of connections from a single IP address.
http_limit_req_module	Limits the request processing rate per a defined key, in particular, the processing rate of requests coming from a single IP address.
http_map_module	Creates variables whose values depend on the values of other variables.
http_memcached_module	Passes requests to a memcached server.
http_proxy_module	Passes HTTP requests to another server.
http_referer_module	Blocks requests with invalid values in the `Referer` header.
http_rewrite_module	Changes the request URI using regular expressions and return redirects; conditionally selects configurations. Requires the PCRE library.
http_scgi_module	Passes requests to an SCGI server.
http_ssi_module	Processes SSI (Server Side Includes) commands in responses passing through it.
http_split_clients_module	Creates variables suitable for A/B testing, also known as split testing.
http_upstream_hash_module	Enables the generic Hash load-balancing method.
http_upstream_ip_hash_module	Enables the IP Hash load-balancing method.
http_upstream_keepalive_module	Enables keepalive connections.
http_upstream_least_conn_module	Enables the Least Connections load-balancing method.
http_upstream_zone_module	Enables shared memory zones.
http_userid_module	Sets cookies suitable for client identification.
http_uwsgi_module	Passes requests to a uwsgi server.

Including Modules Not Built by Default

Many NGINX modules are not built by default, and must be listed on the `configure` command line to be built.

The mail, stream, geoip, image_filter, perl and xslt modules can be compiled as dynamic. See Dynamic Modules for details.

An example of the `configure` command that includes nondefault modules (should be typed as a single line):

```
./configure
--sbin-path=/usr/local/nginx/nginx
--conf-path=/usr/local/nginx/nginx.conf
--pid-path=/usr/local/nginx/nginx.pid
--with-pcre=../pcre2-10.40
--with-zlib=../zlib-1.2.13
--with-http_ssl_module
--with-stream
--with-mail
```

Module Name	Description

- - `--with-cpp_test_module`
  - Tests the C++ compatibility of header files.
- - `--with-debug`
  - Enables the debugging log.
- - `--with-file-aio`
  - Enables asynchronous I/O.
  - Allows using Google Performance tools library.
- - `--` with-http_addition_module
  - Adds text before and after a response.
- - `--` with-http_auth_request_module
  - Implements client authorization based on the result of a subrequest.
- - `--` with-http_dav_module
  - Enables file management automation using the WebDAV protocol.
- - `--with-http_degradation_module`
  - Allows returning an error when a memory size exceeds the defined value.
- - `--` with-http_flv_module
  - Provides pseudo-streaming server-side support for Flash Video (FLV) files.
- - `--` with-http_geoip_module
  - Enables creating variables whose values depend on the client IP address. The module uses MaxMind GeoIP databases. To compile as a separate dynamic module instead, change the option to `--with-http_geoip_module=dynamic`.
- - `--` with-http_gunzip_module
  - Decompresses responses with `Content-Encoding: gzip` for clients that do not support the *zip* encoding method.
- - `--` with-http_gzip_static_module
  - Allows sending precompressed files with the **.gz** filename extension instead of regular files.
- - `--` with-http_image_filter_module
  - Transforms images in JPEG, GIF, and PNG formats. The module requires the LibGD library. To compile as a separate dynamic module instead, change the option to `--with-http_image_filter_module=dynamic`.
- - `--` with-http_mp4_module
  - Provides pseudo-streaming server-side support for MP4 files.
- - `--` with-http_perl_module
  - Used to implement location and variable handlers in Perl and insert Perl calls into SSI. Requires the PERL library. To compile as a separate dynamic module instead, change the option to `--with-http_perl_module=dynamic`.
- - `--` with-http_random_index_module

- Processes requests ending with the slash character ('/') and picks a random file in a directory to serve as an index file.

- - `--` with-http_realip_module
- Changes the client address to the one sent in the specified header field.

- - `--` with-http_secure_link_module
- Used to check authenticity of requested links, protect resources from unauthorized access, and limit link lifetime.

- - `--` with-http_slice_module
- Allows splitting a request into subrequests, each subrequest returns a certain range of response. Provides more effective caching of large files.

- - `--` with-http_ssl_module
- Enables HTTPS support. Requires an SSL library such as OpenSSL.

- - `--` with-http_stub_status_module
- Provides access to basic status information. Note that NGINX Plus customers do not require this module as they are already provided with extended status metrics and interactive dashboard.

- - `--` with-http_sub_module
- Modifies a response by replacing one specified string by another.

- - `--` with-http_xslt_module
- Transforms XML responses using one or more XSLT stylesheets. The module requires the Libxml2 and XSLT libraries. To compile as a separate dynamic module instead, change the option to `--with-http_xslt_module=dynamic`.

- - `--` with-http_v2_module
- Enable support for HTTP/2. See The HTTP/2 Module in NGINX on the NGINX blog for details.

- - `--` with-mail
- Enables mail proxy functionality. To compile as a separate dynamic module instead, change the option to `--with-mail=dynamic`.

- - `--` with-mail_ssl_module
- Provides support for a mail proxy server to work with the SSL/TLS protocol. Requires an SSL library such as OpenSSL.

- - `--` with-stream
- Enables the TCP and UDP proxy functionality. To compile as a separate dynamic module instead, change the option to `--with-stream=dynamic`.

- - `--` with-stream_ssl_module
- Provides support for a stream proxy server to work with the SSL/TLS protocol. Requires an SSL library such as OpenSSL.

- - `--with-threads`
- Enables NGINX to use thread pools. For details, see Thread Pools in NGINX Boost Performance 9x! on the NGINX blog.

Including Third-Party Modules

You can extend NGINX functionality by compiling NGINX Open Source with your own module or a third-party module. Some third-party modules are listed in the NGINX Wiki. Use third-party modules at your own risk as their stability is not guaranteed.

Statically Linked Modules

Most modules built into NGINX Open Source are *statically linked*: they are built into NGINX Open Source at compile time and are linked to the NGINX binary statically. These modules can be disabled only by recompiling NGINX.

To compile NGINX Open Source with a statically linked third-party module, include the `--add-module=<PATH>` option on the `configure` command, where `<PATH>` is the path to the source code (this example is for the RTMP module):

```
./configure ... --add-module=/usr/build/nginx-rtmp-module
```

Dynamically Linked Modules

NGINX modules can also be compiled as a shared object (***.so** file) and then dynamically loaded into NGINX Open Source at runtime. This provides more flexibility, as the module can be loaded or unloaded at any time by adding or removing the associated load_module directive in the NGINX configuration file and reloading the configuration. Note that the module itself must support dynamic linking.

To compile NGINX Open Source with a dynamically loaded third-party module, include the `--add-dynamic-module=<PATH>` option on the `configure` command, where `<PATH>` is the path to the source code:

```
./configure ... --add-dynamic-module=<PATH>
```

The resulting ***.so** files are written to the *prefix***/modules/** directory, where the *prefix* is a directory for server files such as **/usr/local/nginx/**.

To load a dynamic module, add the load_module directive to the NGINX configuration after installation:

```
load_module modules/ngx_mail_module.so;
```

For more information, see Compiling Third-Party Dynamic Modules for NGINX and NGINX Plus on the NGINX blog and Extending NGINX in the Wiki.

Completing the Installation from Source

- Compile and install the build:

  ```
  make
  sudo make install
  ```

- After the installation is finished, start NGINX Open Source:

  ```
  sudo nginx
  ```

NGINX Part of F5

Found a bug? Looking for something new? LET US KNOW

Copyright © F5, Inc. All rights reserved.
Trademarks | Policies | Privacy | California Privacy | Do Not Sell My Personal Information | Cookie Preferences