NGINX High Performance

NGINX is one of the most common free, open source web servers. Its performance-oriented architecture and small footprint makes it an ideal choice for high-traffic websites.

NGINX offers great performance and optimal resource utilization to its administrators. This practical guide walks you through how to tune one of the leading free open source web servers to attain optimal performance for high-traffic sites. It also explores ways to improve network utilization for high loads.

The tour starts with an overview of the NGINX architecture. You will build and configure NGINX for optimal utilization of the hardware available. The book demonstrates various practices to improve last mile content delivery by using timeouts, caching, and compression. You’ll also discover various free open source tools to test and benchmark web server performance, allowing you to verify NGINX performance at every step.

Who this book is written for

System administrators, developers, and engineers looking for ways to achieve maximum performance from NGINX will find this book beneficial. If you are looking for solutions such as how to handle more users from the same system or load your website pages faster, then this is the book for you.

What you will learn from this book

- Compile and run NGINX from source
- Measure NGINX performance and create baselines
- Tweaks for NGINX configuration for last-mile performance
- Work with PHP, Python, and other languages using FCGI
- Learn to integrate with other servers over HTTP
- Set up Memcache for a performance boost
- Fine-tune the TCP stack for improved network utilization
- Construct NGINX extensions for various purposes
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Working with NGINX'
- A synopsis of the book’s content
- More information on NGINX High Performance
Rahul Sharma works as a principal consultant with Xebia India. He has 10 years of experience in building and designing applications on the Java and J2EE platforms. He is an open source enthusiast and has contributed to a variety of open source projects, such as Apache Crunch, Apache Provisionr, Apache HDT, and so on. In his career, he has worked with companies of various sizes, from enterprises to start-ups, and has used NGINX along the way. He often speaks at various technical meetups and shares his knowledge on his personal blog at https://devlearnings.wordpress.com/.
Preface

NGINX is one of the most widely used web servers on the Internet. The server is often named when one is looking to deliver better performance with the same hardware. The server has a state-of-the-art event-based architecture, which enables it to deliver hundreds of thousands of concurrent connections on standard hardware.

As a first step, adopting NGINX leads to better results. However, as with any piece of software, NGINX can also be optimized to serve content faster. This book provides ways to optimize NGINX for last-mile performance. It also aims to provide insights into the NGINX architecture for you to understand it better. The book is not an NGINX learning book and is intended for people with some experience with NGINX. The book only explains those parts of the NGINX configuration that have an impact on performance.

Besides NGINX optimization, the book also talks about the process of benchmarking and about baselines to quantify the gains made. This is an end-to-end book that helps you to tweak the NGINX server's performance.

What this book covers

Chapter 1, Working with NGINX, talks about NGINX's high-performance architecture. It also explains the various modules available in NGINX. The chapter lists details of the amenability of the server to customization. In the end, the chapter builds a simple configuration to deploy example web pages in NGINX.

Chapter 2, Benchmarking the Server, explains performance testing to generate baselines. Siege and JMeter are tools used to measure and benchmark the performance of your server. The chapter aims to generate metrics for the web pages deployed in Chapter 1, Working with NGINX, which can be compared to see performance changes.

Chapter 3, Tweaking NGINX Configuration, covers the parameters: Worker and Worker_process, the use method and multi_accept, Sendfile, directio and aio, tcp_nodely, and tcp_nopush.
Chapter 4, Controlling Buffers, Timeouts, and Compression, lists the features: keepalive, send timeouts, client buffers, gzip compression, and controlling logs.

Chapter 5, Configuring the Network Stack, lists details about tweaking the TCP options to achieve better network utilization. This chapter also talks about various server defaults that need to be customized.

Chapter 6, Using NGINX Cache, shows that NGINX can cache static content, as well as dynamic content. The server provides various directives to cache content. The chapter lists ways to use caches, namely FastCGI Cache, NGINX Proxy Cache, and Memcache, to render content.

Chapter 7, Extending NGINX, provides details of HttpLuaModule, which can be used to extend NGINX for all kinds of activities. The module enables support for Lua, which is a powerful, fast, lightweight, embeddable scripting language. The chapter aims to build an SEO check using Lua.
Igor Sysoev started developing NGINX in 2002. The first version was aimed at delivering static content on web scale. It was released to the public in 2004, thus solving Daniel Kegel’s C10K problem of 10,000 simultaneous connections. NGINX adapted a modular, asynchronous, event-based, nonblocking architecture, which works well to deliver tens of thousands of concurrent connections on a server with typical hardware.

In this chapter, we will cover the following topics:

- The NGINX architecture
- Installing NGINX from source
- Configuring NGINX for HTTP
- Deploying a "Hello world!" page
The NGINX architecture

NGINX has its foundation in event-based architecture (EBA). In EBA, components interact predominantly using event notifications instead of direct method calls. These event notifications, occurring from different tasks, are then queued for processing by an event handler. The event handler runs in an event loop, where it processes an event, de-queues it, and then moves on to the next event. Thus, the work executed by a thread is very similar to that of a scheduler, multiplexing multiple connections to a single flow of execution. The following diagram shows this:

![Event Sources](image)

When compared with the thread-based architecture, EBA gives better performance output. In EBA, there are a fixed number of threads performing tasks and no new threads are formed. Thus, we achieve better utilization of the CPU and an improved memory footprint. There is also no overhead of excessive context switching and no need for a thread stack for each connection. Ideally, the CPU becomes the only apparent bottleneck of an event-driven application.

NGINX runs one master process and several worker processes. The master process reads/evaluates the configuration and maintains the worker processes. All request processing is done by the worker processes. NGINX does not start workers for every request. Instead, it has a fixed pool of workers for request processing. Each worker accepts new requests from a shared listen queue. The worker then uses the available event-notification interfaces, such as epoll and kqueue, to process each connection in an efficient event loop. The idea is to optimize the utilization of the server’s resources using nonblocking/asynchronous mechanisms when possible. By doing so, each worker is able to process thousands of connections. The following diagram shows this:
NGINX has an extensible modular architecture. There is a core module (`ngx_core_module`), which is responsible for connection handling. Then, there are modules for every kind of processing that NGINX offers, for instance an HTTP module (`ngx_http_core_module`) for HTTP processing, an e-mail module (`ngx_mail_core_module`) for e-mail processing, a proxy module (`ngx_http_proxy_module`), and so on. The modular system is quite extensible and allows third-party developers to extend NGINX beyond its existing capabilities. Each worker loads a chain of modules as specified in the NGINX configuration. Every request that a worker handles goes through the loaded chain of modules.

**Installing NGINX from source**

NGINX can be downloaded from [http://nginx.org/](http://nginx.org/). The site provides a stable package and a mainline version package. Both versions are quite stable, and either of them can be used to build NGINX for production environments. The mainline version contains all active development. This essentially means that all new features and noncritical bug fixes are developed here. At times, this may also cause certain third-party modules to break due to a change in the internal APIs. The stable version only gets the critical bug fixes. New features and noncritical bug fixes are not ported to the stable version. NGINX has a well-maintained documentation available at [http://nginx.org/en/docs/](http://nginx.org/en/docs/). The documentation is a great source of information related to NGINX features, internals, and recipes.
Nginx.org runs the released mainline version. You can check this using the following curl command:

```
curl -I http://nginx.org
```

The preceding command will print HTTP headers, which list the server they run on:

```
HTTP/1.1 200 OK
Server: nginx/1.7.7
Content-Type: text/html; charset=utf-8
```

**Build requirements**

Before we start building NGINX, we need to make sure that your system meets the requirements mentioned in the following pages.

**The ANSI C compiler and build system**

As NGINX is written in C, GNU C Compiler (GCC) is recommended to build this. So, make sure you have gcc installed on your box. In addition to this, make sure that PATH contains essential build tools, such as make.

```
$ sudo apt-get install build-essential
```

NGINX provides options to customize/optimize the compilation and linking done by the C compiler. You could also specify a C compiler and preprocessor that is not in PATH:

- **--with-cc**: This specifies an alternative C compiler location to the one in PATH.
- **--with-cpp**: This provides the C preprocessor's location.
- **--with-cc-opt**: This specifies the additional options that are passed to the C compiler. You can pass options to include libraries, for example, `-I /usr/local/include`.
- **--with-ld-opt**: This specifies additional options that are passed to Linker. You can pass options to link libraries, for example, `-L /usr/local/lib`.
- **--with-cpu-opt**: This platform argument can be specified so that a build can be optimized for a specific architecture.
libatomic_ops and AIO – optional requirements

NGINX can use libatomic_ops for memory update operations. The library allows atomic updates, thus removing the lock handling involved in accessing shared memory. You can install the required package and use it with the --with-libatomic configuration parameter. Optionally, you can also download the library from https://github.com/ivmai/libatomic_ops and point to it using the --with-libatomic configuration parameter.

Linux offers asynchronous I/O (AIO). This allows applications to overlap I/O operations with other processing, thus enabling better utilization of resources. NGINX can use this by using the --with-file-aio configuration parameter.

Perl – an optional requirement

NGINX-embedded Perl (ngx_http_perl_module) requires Perl to be installed on your box. The module can be used to build Perl-based configurations in NGINX configuration files. You could also download/build the Perl binary from http://www.perl.org/ and point to it using the --with-perl parameter. Perl modules, used in configurations, can be located with --with-perl modules configuration parameters.

The Perl Compatible Regular Expressions library – an optional requirement

The NGINX HTTP rewrite (ngx_http_rewite_module) module requires the support of the Perl Compatible Regular Expressions (PCRE) library. You could install the package, or you could download the sources from http://www.pcre.org/ and point to it using the --with-pcre configuration parameter. Additional parameters can be passed with the --with-pcre-opt argument; they are passed to the PCRE library.
Install the libpcre3 and libpcre3-dev packages as follows to get the required libraries:

```
$ sudo apt-get install libpcre3 libpcre3-dev
```

**OpenSSL – an optional requirement**

NGINX provides strong cryptography using SSL and TLS protocols. This requires the OpenSSL package to be available on your box. Optionally, you could download the OpenSSL source from http://www.openssl.org/ and point to it using the --with-openssl option. Additional parameters can be passed with the --with-openssl-opt argument; they are passed to the OpenSSL library.

```
$ sudo apt-get install openssl libssl-dev
```

NGINX, by default, does not enable the SSL module. It can be enabled using the --with-http_ssl_module configuration.

**Zlib – an optional requirement**

NGINX can compress HTTP responses in gzip. In order to do this, it requires the support of the Zlib library. You could either install the package or download the sources from http://www.zlib.net/ and point to it using the --with-zlib configuration parameter.

```
$ sudo apt-get install zlib1g zlib1g-dev
```

**Configuring NGINX**

Download and extract the NGINX src package .tar.gz archive from http://nginx.org/en/download.html. Use the following command:

```
$ wget http://nginx.org/download/nginx-1.7.9.tar.gz
```

Next, configure NGINX in the following manner:

```
$ cd nginx-1.7.9
$ ./configure
```
The `configure` command will generate a default NGINX configuration in the form of Makefile. The following output shows the generated configuration of the NGINX binary:

**Configuration summary**
+ using PCRE library: ../pcre-8.35
+ OpenSSL library is not used
+ md5: using system crypto library
+ sha1: using system crypto library
+ using system zlib library

nginx path prefix: "/usr/local/nginx"
nginx binary file: "/usr/local/nginx/sbin/nginx"
nginx configuration prefix: "/usr/local/nginx/conf"
nginx configuration file: "/usr/local/nginx/conf/nginx.conf"
nginx pid file: "/usr/local/nginx/logs/nginx.pid"
nginx error log file: "/usr/local/nginx/logs/error.log"
nginx http access log file: "/usr/local/nginx/logs/access.log"
nginx http client request body temporary files: "client_body_temp"
nginx http proxy temporary files: "proxy_temp"
nginx http fastcgi temporary files: "fastcgi_temp"
nginx http uwsgi temporary files: "uwsgi_temp"
nginx http scgi temporary files: "scgi_temp"

The configuration step provides loads of options to alter default paths and enable/disable all kinds of modules available with NGINX. Using these, you can configure NGINX as per your requirements. This step requires some thought as, once NGINX binaries are built for a configuration, they cannot be altered for default paths or for the support of disabled modules.

## Configuring NGINX defaults

NGINX provides options to change default paths, configuration filenames, log files, and so on. It is not mandatory to provide these options. They have a default value, which is used if the option is not specified:

- `--prefix : /usr/local/nginx`: This specifies the path where NGINX will be installed. All other paths are relative to this location.

- `--sbin-path : prefix/sbin/nginx`: This specifies the name of the NGINX executable binary file.
Working with NGINX

- **--conf-path : prefix/conf/nginx.conf**: NGINX can run a configuration file using the -c filename runtime arguments. If that is not specified, NGINX tries to load a default configuration specified by this option.

- **--pid-path : prefix/logs/nginx.pid**: The NGINX runtime configuration can specify a PID file, the location to store the process ID of the main process. If the pid directive is missing from the configuration file, NGINX stores the information at the location specified by this option.

- **--lock-path : prefix/logs/nginx.lock**: NGINX maintains a lock for shared access to resources. The runtime configuration can specify the lock file. If the lock directive is missing from the configuration file, NGINX stores the information at the location specified by this option.

- **--error-log-path : prefix/logs/error.log**: An error log path can be specified in the NGINX runtime configuration. If the error_log directive is missing in the specified configuration, then NGINX writes the log at the location specified by this option.

- **--http-log-path : prefix/logs/access.log**: The HTTP-access log paths can be specified in the NGINX runtime configuration. If the access_log directive is missing in the specified configuration, NGINX writes logs at the location specified by this option.

- **--with-debug**: This option enables a detailed debug log in NGINX. This option is not enabled by default.

- **--user : nobody**: The NGINX runtime configuration can specify an unprivileged user run NGINX worker processes.

- **--group : nobody**: The NGINX runtime configuration can specify a group that will own NGINX worker processes.

- **--build**: This option assigns a name to the generated binary. The name would be available in the NGINX -v command.

**Configuring NGINX modules**

NGINX has an extensive set of features. It can be used as a web server, a web cache, a load balancer, an e-mail proxy, and so on. All these features of NGINX are compiled and configured as modules. Certain modules are enabled by default, while some need to be enabled. Modules should be carefully selected when building a high-performance system to keep a low memory footprint.
Chapter 1

Configuring NGINX for the Web

NGINX, by default, installs the HTTP module. The HTTP server can be disabled using the `--without-http` configuration parameter. Along with the HTTP module, NGINX also enables the following modules by default. The modules can be disabled selectively using the specified configuration parameter. Here is a list of a few parameters:

- `--without-http_access_module`: This module allows us to control access from a limited set of IP addresses.
- `--without-http_autoindex_module`: This module builds the index file, which can generate a directory listing for requests ending with a forward slash (`/`).
- `--without-http_auth_basic_module`: This module allows access control using the basic authentication protocol.
- `--without-http_browser_module`: This module enables us to create variables based on the value of the `User-Agent` request header field.
- `--without-http_charset_module`: This module enables the conversion of the response data between different character encodings and the setting of the `Content-Type` response header field.
- `--without-http_empty_gif_module`: This module allows NGINX to serve a 1 x 1 transparent GIF from memory.
- `--without-http_fastcgi_module`: This module allows NGINX to send requests to the FastCGI server.
- `--without-http_geo_module`: This module allows us to set up configuration variables based on the client IP address. The variables can then be used in other modules for certain actions.
- `--without-http_gzip_module`: This module allows us to compress responses in the gzip format, thus reducing the amount of bytes transferred.
- `--without-http-cache`: NGINX is capable of using the HTTP cache, that is, setting the correct request headers to enable content caching. This can be used when NGINX is set up as a proxy to an upstream content provider. The option can disable the cache, but it is not advisable as the cache is quite handy in most setups.
- `--without-http_limit_conn_module`: This module allows us to control the number of concurrent connections from a single IP. Post limits, it sends the 503 error response.

---

[9]


- **--without-http_limit_req_module:** This module allows us to define the request processing rate for particular keys. It keeps track of requests from a single IP, queuing them if the request rate exceeds the defined processing rate. After the maximum queue size has been reached, it sends a **503 error** response.

- **--without-http_map_module:** This module allows us to set up configuration variables that derive value from other existing variables.

- **--without-http_memcached_module:** This module allows NGINX to serve requests directly from a memcached server. The memcached server should contain the response in advance by means external to NGINX.

- **--without-http_proxy_module:** This module enables NGINX to send requests to other servers.

- **--without-http_referer_module:** This module allows NGINX to filter requests based on the values in the **Referrer** header field.

- **--without-http_rewrite_module:** This module allows NGINX to manipulate URLs, using regular expressions, based on certain conditions.

- **--without-http_scgi_module:** This module allows NGINX to send requests to the SCGI server.

- **--without-http_split_clients_module:** This module allows NGINX to configure variables for A/B testing.

- **--without-http_ssl_module:** This module enables NGINX to process SSI commands

- **--without-http_upstream_ip_hash_module --without-http_upstream_keepalive_module --without-http_upstream_least_conn_module**

These modules enable load-balancing capabilities in NGINX. A number of servers can be grouped together to provide upstream content to NGINX. The upstream group can distribute requests between them using any of the following methods:

- client IP address hash
- keep-alive parameters
- number of connected clients

NGINX has separate modules depending on the method selected for request distribution.
• **--without-http_userid_module**
  This module allows NGINX to use cookies for client identification.

• **--without-http_uwsgi_module**
  This module allows NGINX to send requests to the uWSGI server.

Besides the preceding modules, which are enabled by default, the NGINX catalog has modules that can offer a range of features, such as SSL support, streaming media, and so on. These modules are disabled by default and need to be enabled at compile time. The following is the list of modules that can be enabled at compile time.

• **--with-http_addition_module**: This module can modify the response returned for a given request. It adds a response from another subrequest before and after the actual response.

• **--with-http_auth_request_module**: The module can enable client authorization based on the subrequest. The client accesses protected resources, which trigger a subrequest for authorization. Depending on the response code received, access is allowed (2xx) or denied (401 error/403 error).

• **--with-http_degradation_module**: This module allows us to return either the 444 error or the 204 error in low-memory conditions. This module is not available on Linux platforms.

• **--with-http_perl_module**: This module allows you to insert Perl code into your NGINX configuration files and to make Perl calls from SSI.

• **--with-http_flv_module**: This module provides support to stream Flash media files.

• **--with-http_geoip_module**: This module allows us to use Maxmind GeoIP-Databases to build custom variables based on client IP addresses.

  Note that in order to use this module, you need to have the Maxmind library in your path.

• **--with-http_google_perftools_module**: This module allows you to use Google's performance tools (http://code.google.com/p/gperftools/) to profile NGINX workers.
• **--with-http_gzip_static_module**: This module enables NGINX to serve precompressed static content in the .gz form.

• **--with-http_image_filter_module**: This module allows image transformation using the libgd library.

  Note that in order to use this module, you need to have the libgd library in your PATH.

• **--with-http_mp4_module**: This module enables the streaming of MP4 files.

• **--with-http_random_index_module**: This module enables NGINX to randomly select and serve any file in a directory as the index file.

• **--with-http_realip_module**: This module can be used to correctly determine the client IP. If NGINX is used behind a proxy or a load balancer, then the module can extract the client IP from the correct header field.

• **--with-http_secure_link_module**: This module is used to access control of locations by generating checksums.

• **--with-http_spdy_module**: This module enables support for the SPDY protocol.

• **--with-http_ssl_module**: This module enables HTTPS support in NGINX.

• **--with-http_stub_status_module**: This module builds basic server statistics.

• **--with-http_sub_module**: This module modifies the generated response by replacing certain search strings with replacement strings.

• **--with-http_dav_module**: This module extends support for the WebDAV protocol over HTTP.

• **--with-http_xslt_module**: This module modifies the generated response using XSLT transformations.

  Note that you need to have the libxml2 and libxslt libraries in your PATH.
Configuring NGINX for e-mail

NGINX has the capability to serve as an e-mail proxy. It supports POP3, IMAP, and SMTP protocols along with SSL capabilities. The e-mail modules are disabled by default. They need to be configured with the following parameters:

- **--with-mail**: This module enables e-mail capabilities. It enables POP3, IMAP, and SMTP modules.
- **--with-mail_ssl_module**: This module enables the support for the SSL/TLS protocols for e-mail services.
- **--without-mail_pop3_module**: This module enables the support for the POP3 protocol. The module is enabled by the e-mail module.
- **--without-mail_imap_module**: This module enables the support for the IMAP protocol. This module is enabled by the e-mail module.
- **--without-mail_smtp_module**: This module enables the support for the SMTP protocol. The module is enabled by the e-mail module.

Configuring third-party modules

NGINX is completely modular in nature, which essentially means that you can build your own modules to extend its existing capabilities. The NGINX community has built a large number of third-party modules that can perform various tasks. If you want to use a few of these modules, you need to compile them with the NGINX source. Download the module source and point to the source using the **--add-module** configure parameter. You can add as many modules as you want.

The NGINX official release does not support these modules, so make sure that you test the binaries before setting them up for production. If you are building from the mainline source, then modules relying on internal APIs can break while moving from one version to another.

NGINX – the complete package

Now that you know how to configure the available modules in NGINX, you may want to customize the binary as per your needs. While you are enabling/disabling modules, do make a note of all the libraries that NGINX will rely on. For the modules available from the NGINX catalog, it may be a good idea to download the required sources and point to them using correct parameters. The following command configures an NGINX binary:

```
$ ./configure \
   --prefix=/etc/nginx \\n```
The following output summary lists the libraries used and the NGINX defaults generated. Makefile, which is generated as a result of the preceding code, can be used to build and install the NGINX binary. Here’s the output summary of the preceding command:

Configuration summary
+ using PCRE library: ../pcre-8.36
+ using OpenSSL library: ../openssl-1.0.2a
+ md5: using OpenSSL library
+ sha1: using OpenSSL library
+ using system zlib library
+ using libatomic_ops library: ../libatomic_ops-7.2

nginx path prefix: "/etc/nginx"
nginx binary file: "/etc/nginx/sbin/nginx"
nginx configuration prefix: "/etc/nginx/conf"
nginx configuration file: "/etc/nginx/conf/nginx.conf"
nginx pid file: "/etc/nginx/logs/nginx.pid"
nginx error log file: "/etc/nginx/logs/error.log"
nginx http access log file: "/etc/nginx/logs/access.log"
nginx http client request body temporary files: "client_body_temp"
nginx http proxy temporary files: "proxy_temp"
Building and installing NGINX

The configure command generated Makefile for the specified configuration. Makefile can now be used to install NGINX in the following manner:

```
$ make
$ sudo make install
```

The preceding command shows the output listing the locations of installation:

```
test -d '/etc/nginx' || mkdir -p '/etc/nginx'
test -d '/etc/nginx/sbin' || mkdir -p '/etc/nginx/sbin'
test ! -f '/etc/nginx/sbin/nginx' || mv '/etc/nginx/sbin/nginx' '/etc/nginx/sbin/nginx.old'
cp objs/nginx '/etc/nginx/sbin/nginx'
test -d '/etc/nginx/conf' || mkdir -p '/etc/nginx/conf'
cp conf/koi-win '/etc/nginx/conf'
cp conf/koi-utf '/etc/nginx/conf'
cp conf/win-utf '/etc/nginx/conf'
test -f '/etc/nginx/conf/mime.types' || cp conf/mime.types '/etc/nginx/conf'
cp conf/mime.types '/etc/nginx/conf/mime.types.default'
test -f '/etc/nginx/conf/fastcgi_params' || cp conf/fastcgi_params '/etc/nginx/conf'
cp conf/fastcgi_params '/etc/nginx/conf/fastcgi_params.default'
test -f '/etc/nginx/conf/fastcgi.conf' || cp conf/fastcgi.conf '/etc/nginx/conf'
cp conf/fastcgi.conf '/etc/nginx/conf'
```

The make command prints lot of debugging information, for example, the library paths, the default configuration paths, and so on. You can run it with the -s option to disable all the information.

After installation, NGINX is available in the specified prefix directory. You can go to the specified location and run it with the sudo command. Alternatively, you can add it to PATH using update-alternatives and then run it. The following command shows this:

```
$ sudo update-alternatives --install /usr/bin/nginx nginx
/etc/nginx/sbin/nginx 1
$ sudo nginx
```
You can check this by loading http://localhost in your browser. It should display the following NGINX page:

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Deploying in NGINX

Now that we have successfully installed NGINX, we want to try out some deployments in NGINX.

In this section, we have a Hello world! web page that we will deploy in NGINX. The code uses the Bootstrap library, which needs to be packaged with the code. The complete package has the following structure:

```
Code
  └── index.html
    └── css
        └── bootstrap.min.css
```
The following is the HTML markup of index.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Using Nginx</title>
  <link href="css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <div class="container">
    <div class="jumbotron">
      <h1>Hello world!</h1>
      <p>Deploying in Nginx</p>
    </div>
  </div>
</body>
</html>
```

In this HTML markup, we will display the heading "Hello, world!" and a bit of text on the web page, which has the title Using Nginx. The Bootstrap CSS has been used to style the content. You can download it from http://getbootstrap.com/getting-started/ or use the wget command from a CDN and add it to the css folder:

```bash
$ wget https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css
```

Details of the HTML source code will not be covered here as they are beyond the scope of the book. The purpose of this code is to deploy a simple web page in NGINX.

**Deploying NGINX**

We want to load the page created previously at the location http://localhost/hello. In order to do so, we need to write a configuration block using the following directives:

- location: This defines a configuration for a URI. The location URI can be a prefix or a regular expression.
- alias: This defines the directory path for the specified location directive. This is the location from where all the files are served.
- include: This allows the inclusion of the configuration blocks defined in one file in another file.
NGINX, by default, loads the deployment from `nginx.conf` in the installation directory.

In the preceding section, NGINX was installed in `/etc/nginx`, thus `nginx.conf` should be available in `/etc/nginx/conf/nginx.conf`.

NGINX allows us to write configurations across multiple files as logical units. We can include these files in a main file to define the complete configuration. Rather than writing configuration blocks in `nginx.conf`, we will create new files and then include them in the required locations in `nginx.conf`.

For our purposes, we will create `nginx-localhost-server.conf` in the `/etc/nginx/conf` directory with the following configuration block:

```nginx
location /hello {
  alias "${ABSOLUTE_PATH_TO_CODE}";
}
```

Here, we have defined the `/hello` location and set it to serve from `index.html` at `$ABSOLUTE_PATH_TO_CODE`. When the page is loaded, it tries to load the CSS from `/hello/css/bootstrap.min.css`. The location directive successfully matches the `/hello` prefix and serves the CSS from the `$ABSOLUTE_PATH_TO_CODE/css` path.

The `/hello` prefix needs to available in the localhost server's name; thus, we need to include `nginx-localhost-server.conf` in the block, which defines the localhost server in `nginx.conf`. The following code shows this:

```nginx
server_name localhost;

# include the hello location
include nginx-localhost-server.conf;

location / {
  root html;
  index index.html index.htm;
}
```

Now, all that is left for us to do is reload NGINX. It is always a good idea to test the NGINX configuration before loading. Testing can flag up possible errors arising from invalid configurations, which would be discovered while loading NGINX. Use the `-t` switch to test the configuration, as shown in the following code:

```sh
$ sudo nginx -t
```
If the test is successful, the command will print the following output; otherwise, it will print the errors found, if any:

```
nginx: the configuration file /etc/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/conf/nginx.conf test is successful
```

Once we have verified the configuration, reload NGINX with the following configuration:

```
$ sudo nginx -s reload
```

Verify the page at `http://localhost/hello`. It should show the following content:
Summary

This chapter gave insights into the components of the performance-oriented NGINX architecture and the EBA model. NGINX was compiled from source with components that suit our requirements. The custom build package was installed and a sample web page was deployed to it.

The purpose of this chapter was to give you a crash course in NGINX. In the subsequent chapters, we will see ways to measure your NGINX website's performance and optimize it for the last mile.
Where to buy this book

You can buy NGINX High Performance from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.