Nginx: the High-Performance Web Server and Reverse Proxy

HOW-TOs (/tag/how-tos)

by Will Reese on September 1, 2008

Apache is the most popular Web server and one of the most successful open-source projects of all time. Since April 1996, Apache has served more Web sites than any other Web server. Many of the world's largest Web sites, including YouTube, Facebook, Wikipedia and Craigslist, use Apache to serve billions of page views per month. Over the years, Apache has proven itself to be a very stable, secure and configurable Web server. Although

You May Like

SFTP Port Forwardinc Enabling

Apache is an excellent Web server, what if there were an alternative with the same functionality, a simpler configuration and better performance? That Web server exists, and it's called Nginx.

Nginx, pronounced “Engine X”, is a high-performance Web server and reverse proxy. It was created by Igor Sysoev for www.rambler.ru (http://www.rambler.ru), Russia's second-largest Web site. Rambler has used Nginx since summer 2004, and it's currently serving about 500 million requests per day. Like Apache, Nginx is used by some of the largest Web sites in the US, including WordPress (#26), YouPorn (#27), Hulu and MochiMedia. As of May 2008, Nginx is the fourth-most-popular Web server, and it is currently serving more than two million Web sites. As it is only trailing behind Apache, IIS and GFE, it is effectively the second-most-popular Web server available for Linux.

Why Use Nginx?

Like Apache, Nginx has all the features you would expect from a leading Web server:

- Static file serving.
- SSL/TLS support.
- Virtual hosts.
- Reverse proxying.
- Load balancing.
- Compression.
- Access controls.
- URL rewriting.
- Custom logging.

(/content/sftp-port-forwarding-enabling-suppressed-functionality)

Suppresse
Functionali
(/content/sf
port-
forwarding-
enabling-
suppressec
functionalit

Charles Fisher (/users/char fisher)

- Server-side includes.
- WebDAV.
- FLV streaming.
- FastCGI.

It is stable, secure and very easy to configure, as you will see later in the article. However, the main advantages of Nginx over Apache are performance and efficiency.

I ran a simple test against Nginx v0.5.22 and Apache v2.2.8 using ab (Apache's benchmarking tool). During the tests, I monitored the system with vmstat and top. The results indicate that Nginx outperforms Apache when serving static content. Both servers performed best with a concurrency of 100. Apache used four worker processes (threaded mode), 30% CPU and 17MB of memory to serve 6,500 requests per second. Nginx used one worker, 15% CPU and 1MB of memory to serve 11,500 requests per second.

Nginx is able to serve more requests per second with less resources because of its architecture. It consists of a master process, which delegates work to one or more worker processes. Each worker handles multiple requests in an event-driven or asynchronous manner using special functionality from the Linux kernel (epoll/select/poll). This allows Nginx to handle a large number of concurrent requests quickly with very little overhead. Apache can be configured to use either a process per request (pre-fork) or a thread for each request (worker). Although Apache's

(/content/fault-tolerant-sftp-scripting-retry-failed-transfers-automatically)
Fault-Tolerant SFTP scripting - Retry Failed Transfers Automatica (/content/fa tolerant-sftp-scripting-retry-failed-transfers-automatica

Charles Fisher (/users/char fisher)

threaded mode performs much better than its pre-fork mode, it still uses more memory and CPU than Nginx's event-driven architecture.

Installation and Basic Usage

Nginx is available in most Linux distributions. For this article, I use Ubuntu 8.04 (Hardy), which includes Nginx version 0.5.33. If your distro does not have Nginx, or if you want to run a newer version, you always can download the latest stable version (v0.6.31 at the time of this writing) and install from source.

Run the following command as root to install Nginx:

```
# apt-get install nginx
```

Now that Nginx is installed, you can use the startup script to start, stop or restart the Web server:

```
# /etc/init.d/nginx start
# /etc/init.d/nginx stop
# /etc/init.d/nginx restart
```

Most configuration changes do not require a restart, in which case you can use the reload command. It is generally a good idea to test the Nginx configuration file for errors before reloading:

```
# nginx -t
# /etc/init.d/nginx reload
```

(/content/installing-libreoffice-slackware-15)
Installing LibreOffice On Slackware 15 (/content/in libreoffice-slackware-15)
Terrell Prude' Jr. (/users/terre prude-jr)

(/content/sqlite-secrecy-management-tools-and-methods)
SQLite for Secrecy Manageme - Tools and Methods (/content/s secrecy-

Let's go ahead and start the server:

```
# /etc/init.d/nginx start
```

Nginx now should be running on your machine. If you open http://127.0.0.1/ in your browser, you should see a page with “Welcome to nginx!”.

Main Configuration File

Now that Nginx is installed, let's take a look at its config file, located at /etc/nginx/nginx.conf. This file contains the server-wide settings for Nginx, and it should look similar to this:

managem
tools-and-
methods)

Charles Fisher (/users/char fisher)

```
user www-data;
worker_processes  1;
error_log  /var/log/nginx/error.log;
pid  /var/run/nginx.pid;
events {
  worker_connections  1024;
}
http {
  include  /etc/nginx/mime.types;
  default_type  application/octet-stream;
  access_log  /var/log/nginx/access.log;
  sendfile  on;
  keepalive_timeout  65;
  tcp_nodelay  on;
  gzip  on;
  include /etc/nginx/sites-enabled/*;
}
```

We are not going to change any of these settings, but let's talk about some of them to help us understand how Nginx works. The worker_processes setting tells Nginx how many child processes to start. If your server has more than one processor or is performing large amounts of disk IO, you might want to try increasing this number to see if you get better performance. The worker_connections setting limits the number of concurrent connections per worker process. To determine the maximum number of concurrent requests, you simply multiply worker_processes by worker_connections.

The error_log and access_log settings indicate the default logging locations. You also can configure these settings on a per-site basis, as you will see later in the article. Like Apache, Nginx is configured to run as the www-data user, but you easily can change this with the user setting. The startup script for Nginx needs to know the process ID for the master process, which is stored in /var/run/nginx.pid, as indicated by the pid setting.

The sendfile setting allows Nginx to use a special Linux system call to send a file over the network in a very efficient manner. The gzip option instructs Nginx to compress each response, which uses more CPU but saves bandwidth and decreases response time. Additionally, Nginx provides another compression module called gzip precompression (available as of version 0.6.24). This module looks for a compressed copy of the file with a .gz extension in the same location and serves it to gzip-enabled clients. This prevents having to compress the file each time it's requested.

The last setting we are concerned with is the include directive for the sites-enabled directory. Inside /etc/nginx, you'll see two other directories, /etc/nginx/sites-available and /etc/nginx/sites-enabled. For each Web site you want to host with Nginx, you should create a config file in /etc/nginx/sites-available, then create a symlink in /etc/nginx/sites-enabled that points to the config file you created. The main Nginx config file includes all the files in /etc/nginx/sites-enabled. This helps organize your configuration files and makes it very easy to enable and disable specific Web sites.

Static Web Server

Now that we covered the main configuration file, let's create a config file for a basic Web site. Before we begin, we need to disable the default site that Ubuntu created for us:

```
# rm -f /etc/nginx/sites-enabled/default
```

Now, create a new configuration file called /etc/nginx/sites-available/basic with the following contents:

```
server {
  listen  127.0.0.1:80;
  server_name  basic;
  access_log  /var/log/nginx/basic.access.log;
  error_log  /var/log/nginx/basic.error.log;
  location  / {
    root  /var/www/basic;
    index  index.html index.htm;
  }
}
```

Create the root directory and index.html file:

```
# mkdir /var/www/basic
# cd /var/www/basic
# echo "Basic Web Site" > index.html
```

Enable the site and restart Nginx:

```
# cd /etc/nginx/sites-enabled
# ln -s ../sites-available/basic .
# /etc/init.d/nginx restart
```

If you open http://127.0.0.1/ in your browser, you should see a page with “Basic Web Site”. As you can see, it is very easy to create a new site using Nginx.

Let's go over the new configuration file we created. The server directive is used to define a new virtual server, and all of its settings are enclosed in braces. The listen directive indicates the IP and port on which this server will accept requests, and server_name sets the hostname for your virtual server. As I mentioned earlier, the access_log and error_log settings can be set on a per-site basis. It is usually a good idea to provide each site with its own set of log files.

Next is the location directive, which allows you to modify the settings for different parts of your site. In our case, we have only one location for the entire site. However, you can have multiple location directives, and you can use regular expressions to define them. We have two other directives inside our location block: root and index. The root directive is used to define the document root for this location. This means a request for /img/test.gif would look for the file /var/www/localhost/img/test.gif. Finally, the index directive tells Nginx what files to use as the default file for this location.
Static Web Server with SSL

Some Web sites, such as on-line stores, require secure communication (HTTPS) to protect credit-card transactions and customer information. Like Apache, Nginx supports HTTPS via an SSL module, and it's very easy to set up.

First, you need to generate an SSL certificate. The openssl command will ask you a bunch of questions, but you simply can press Enter for each one:

```
# apt-get install openssl
# mkdir /etc/nginx/ssl
# cd /etc/nginx/ssl
# openssl req -new -x509 -nodes -out server.crt -keyout
```

Create a new config file called /etc/nginx/sites-available/secure, which contains the following:

```
server {
  listen   127.0.0.1:443;
  server_name  secure;
  access_log  /var/log/nginx/secure.access.log;
  error_log  /var/log/nginx/secure.error.log;
  ssl on;
  ssl_certificate /etc/nginx/ssl/server.crt;
  ssl_certificate_key /etc/nginx/ssl/server.key;
  location / {
    root   /var/www/secure;
    index  index.html index.htm;
  }
}
```

Create the root directory and index.html file:

```
# mkdir /var/www/secure
# cd /var/www/secure
# echo "Secure Web Site" > index.html
```

Enable the site and restart Nginx:

```
# cd /etc/nginx/sites-enabled
# ln -s ../sites-available/secure .
# /etc/init.d/nginx restart
```

If you open https://127.0.0.1/ in your browser (note the https), you probably will get a warning about not being able to verify the certificate. That's because we are using a self-signed certificate for this example. Go ahead and tell your browser to accept the certificate, and you should see a page with “Secure Web Site”.

This config file is very similar to our previous config, but there are a few differences. First, notice that this new server is listening on port 443, which is the standard port for HTTPS. Second, we enabled the SSL module with the line `ssl on;`. If you compiled Nginx yourself instead of using the Ubuntu package, you need to make sure you specified `--with-http_ssl_module` when you ran `./configure`; otherwise, the SSL module will not be available. Third, we used the ssl_certificate and ssl_certificate_key directives to point to the certificate and key we created earlier.

Virtual Hosting

In many cases, you will want to run multiple Web sites from a single server. This is called virtual hosting, and Nginx supports both IP- and name-based vhosts.

Let's create two virtual hosts: one.example.com and two.example.com. First, we need to add a line to our /etc/hosts file, so that one.example.com and two.example.com point to our server (normally you would do this using DNS):

```
# echo "127.0.0.1 one.example.com two.example.com" >> /e
```

Now, we need to create a configuration file for each site. First, create a file called /etc/nginx/sites-available/one with the following contents:

```
server {
  listen   127.0.0.1:80;
  server_name  one.example.com;
  access_log  /var/log/nginx/one.access.log;
  error_log /var/log/nginx/one.error.log;
  location / {
    root   /var/www/one;
    index  index.html index.htm;
  }
}
```

Then, make a copy of that file called /etc/nginx/sites-available/two, and replace each occurrence of “one” with “two”:

```
# cd /etc/nginx/sites-available
# cp one two
# sed -i "s/one/two/" two
```

Create the root directories and index.html files:

```
# mkdir /var/www/{one,two}
# echo "Site 1" > /var/www/one/index.html
# echo "Site 2" > /var/www/two/index.html
```

Enable the sites and restart Nginx:

```
# cd /etc/nginx/sites-enabled
# ln -s ../sites-available/one .
# ln -s ../sites-available/two .
# /etc/init.d/nginx restart
```

If you open http://one.example.com/ in your browser, you should see a page with “Site 1”. For http://two.example.com/, you should see “Site 2”.

We just created two name-based virtual hosts running on 127.0.0.1 by changing the server_name directive. For IP-based virtual hosts, simply change the listen directive to use a different IP for each site.

Now, go ahead and disable these two virtual hosts:

```
# rm -f /etc/nginx/sites-enabled/one
# rm -f /etc/nginx/sites-enabled/two
# /etc/init.d/nginx restart
```

Don't forget to remove the line we added to /etc/hosts when you are done.

Reverse Proxy and Load Balancer

In addition to being an extremely fast static Web server, Nginx also is a load balancer and reverse proxy. A load balancer is a device used to spread work out across multiple servers or processes, and a reverse proxy is a server that transparently hands off requests to another server. Among other things, this allows Nginx to handle requests for static content and to load-balance requests for dynamic content across many different back-end servers or processes.

For this example, let's create a very simple Python Web server to serve up some dynamic content. Don't worry if you are not familiar with Python; we're just using it to display a Web page that indicates on which port the server is running. Save the following to a file called /tmp/server.py:

```
import sys,BaseHTTPServer as B
class Handler(B.BaseHTTPRequestHandler):
  def do_GET(self):
    self.wfile.write("Served from port %s" % port)
  def log_message(self, *args):
    pass
if __name__ == '__main__':
  host,port = sys.argv[1:3]
  server = B.HTTPServer((host,int(port)), Handler)
  server.serve_forever()
```

Now we can start two of these local servers, each on a different port:

```
# python /tmp/server.py 127.0.0.1 8001 &
# python /tmp/server.py 127.0.0.1 8002 &
```

If you open http://127.0.0.1:8001/ in your browser, you should see “Served from port 8001”, and if you open http://127.0.0.1:8002/, you should see “Served from port 8002”.

Now, create a new configuration file called /etc/nginx/sites-available/proxy with the following contents:

```
upstream python_servers {
  server 127.0.0.1:8001;
  server 127.0.0.1:8002;
}
server {
  listen   127.0.0.1:8000;
  server_name  proxy;
  access_log  /var/log/nginx/proxy.access.log;
  error_log /var/log/nginx/proxy.error.log;
  location / {
    proxy_pass http://python_servers;
  }
}
```

Enable the site and restart Nginx:

```
# cd /etc/nginx/sites-enabled
# ln -s ../sites-available/proxy .
# /etc/init.d/nginx restart
```

If you open http://127.0.0.1:8000/ in your browser, you should see a page with either “Served from port 8001” or “Served from port 8002”, and it should alternate each time you refresh the page.

Let's go over some of these new settings. The upstream block defines a name for a group of back-end servers. In our case, we defined a group named python_servers, which contains the two local Python servers we started on port 8001 and 8002. We then configured Nginx to hand off all requests to our back-end servers with the line `proxy_pass http://python_servers;`. Nginx automatically load-balances the requests to each Python server in a round-robin manner. You also can set weights for each back end, so you can direct more or fewer requests to specific servers.

Nginx handles back-end failures automatically and will stop sending requests to a failed back-end server until it starts responding again. To demonstrate this, we can kill off the Python process that's running on port 8001. Use the jobs command to find the job number for the Python process running on port 8001, and use `kill %<job number>` to kill the process:

```
# jobs
# kill %1
```

Open http://127.0.0.1:8000/ in your browser and keep refreshing the page, you should see only the “Served from port 8002” page. Nginx detected that the back-end server from port 8001 was not responding, so it stopped sending requests to that server. Now, restart the Python process for port 8001:

```
# python /tmp/server.py 127.0.0.1 8001 &
```

Keep refreshing the page and you should see your browser start alternating between “Served from port 8001” and “Served from port 8002” again. Nginx detected that the port 8001 back end was responding and began sending requests to it.

Conclusion

Whether you are looking to get the most out of your VPS or are attempting to scale one of the largest Web sites in the world, Nginx may be the best tool for the job. It's fast, stable and easy to use. Thanks to Igor Sysoev for creating this excellent piece of software.

Resources

Nginx Web Site: wiki.codemongers.com/Main (http://wiki.codemongers.com/Main)

Module Comparison Index: wiki.codemongers.com/NginxModuleComparisonMatrix (http://wiki.codemongers.com/NginxModuleComparisonMatrix)

Testimonials: wiki.codemongers.com/NginxWhyUseIt (http://wiki.codemongers.com/NginxWhyUseIt)

Nginx at WordPress: barry.wordpress.com/2008/04/28/load-balancer-update (http://barry.wordpress.com/2008/04/28/load-balancer-update)

Facebook App Using Nginx: highscalability.com/friends-sale-architecture-300-million-page-view-month-facebook-ror-app (http://highscalability.com/friends-sale-architecture-300-million-page-view-month-facebook-ror-app)
Will Reese has worked with Linux for the past ten years, primarily scaling Web applications running on Apache, Python and PostgreSQL. He enjoys beating Cory Wright at foosball and Wii Tennis.

Load 1 comment (https://www.linuxjournal.com/article/10108#disqus_thread)