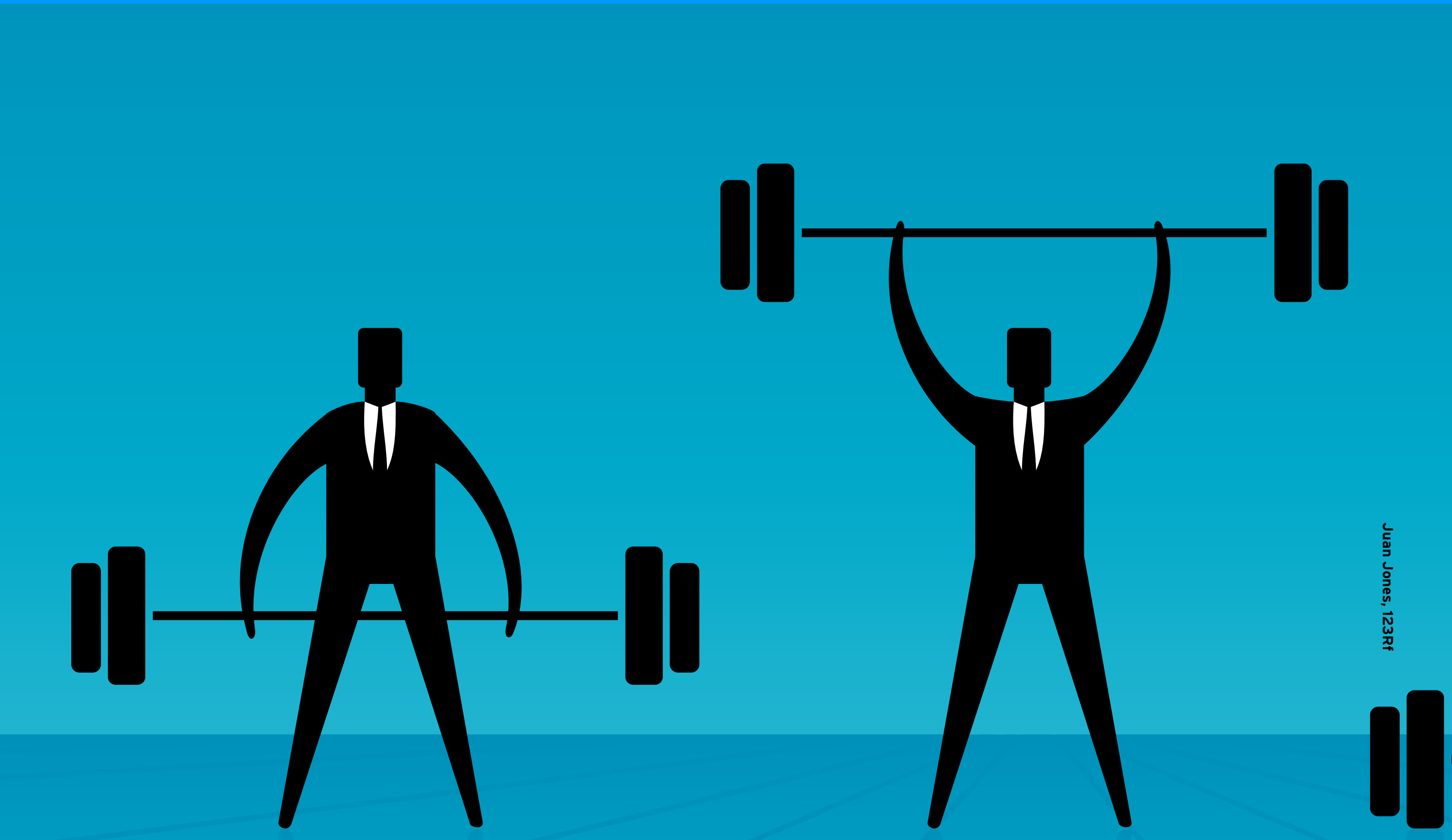


Web service and reverse proxy with the speedy nginx

SMALL BUT POWERFUL

The fast and practical Nginx web server is easy to configure and extend. **BY JEREMY GARCIA**

Nginx (pronounced Engine-ex) is an Open Source HTTP server and reverse proxy. The Nginx web server is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption. Nginx, which was written by Igor Sysoev, is used with many high-profile sites, including WordPress.com, Hulu, and LinuxQuestions.org. In addition to its HTTP-related functionality, nginx can also serve as an IMAP/POP3 proxy.

Nginx's basic HTTP support includes the ability to serve static files, accelerated reverse proxying with optional caching, simple load balancing and fault tolerance, remote FastCGI with caching/acceleration, and SSL/TLS server name indication (SNI). Like Apache's httpd, nginx has many features that are implemented in a modular fashion and only need to be enabled if you plan to use them. Unlike the process-based httpd, though, nginx is asynchronous. The main advantage of the asynchronous approach is scalability. In a process-based server, each simultaneous connection requires a thread, which can lead to significant overhead, especially when under high load. An asynchronous server, on the other hand, is event-driven and handles requests in a single (or at least very few) threads.

Installation

If you have a simple site or are starting from scratch, it's very possible you can completely forgo Apache and just use nginx for all your HTTP needs. If you choose to go this route, I recommend you visit the nginx Modules page [2] and verify that all the functionality you require is available with an nginx module. This article describes the common case of nginx acting as a load balancing reverse proxy to multiple existing Apache httpd back-ends. Nginx will serve some static content and then automatically gzip the dynamic content it is proxying from httpd.

Most Linux distributions have nginx in their package repositories, so you can easily install it using the standard package manager. If the packaged version for your distro is outdated or not available, visit *http://nginx.net/* and download the latest stable version (0.7.61 at the time this article was written). The installation involves the standard *./configure && make && make install* process. Although the defaults should work in most cases, I recommend you check out the available configuration options and tailor them to suit your environment. By default, you

Table 1: Layout

machine	Front-end IP	Back-end IP
nginx	10.0.0.1	192.168.1.1
web01	none	192.168.1.2
web02	none	192.168.1.3

should see something similar to the following after you run *./configure*:

```
Configuration summary
  + using system PCRE library
  + using system OpenSSL library
  + md5: using OpenSSL library
  + using sha1 library: ↩
     /usr/include
  + using system zlib library
```

It's important to verify that the preceding libraries are found if you plan on using any of the functionality associated with them. For instance, the *Rewrite* module requires the PCRE library, and SSL support requires OpenSSL. With nginx installed, it's time set up a basic configuration.

Basic Configuration

This article assumes a three-server infrastructure. The machine with nginx should ideally be on both a front-end public facing network and a back-end private network. The machines running httpd do not need any front-end network access. The layout for this article is shown in Table 1. With Table 1 in mind, edit the *nginx.conf* file with the information shown in Listing 1.

This configuration will result in both back-end machines getting the same number of requests. By default, nginx performs simple per-request, round-robin load balancing. If you'd like requests to be distributed between upstreams based on the IP address of the client, you can use the *ip_hash* directive. Additional, more advanced load balancing algorithm support is planned for a future nginx release. Note that, by default, all requests passed to the back-end httpd processes will appear to originate from the IP address of the nginx machine. I suggest you pass the IP address from the original request to httpd via the *X-Forwarded-For HTTP* header and then intercept that information with the httpd *mod_rpaf* module, which will change the remote IP address visible to other httpd modules. The *mod_rpaf* module is open source and available for download [3].

SSL Support

If you're using nginx as a load balancing reverse proxy, configuring it to handle SSL requests has multiple benefits. This approach simplifies your httpd configuration, offloads the CPU-load associated with SSL processing, and allows for easier load balancing, since it resolves the need to have "sticky" SSL sessions. Configuring SSL support is simple and requires the same CRT and KEY files as an httpd SSL configuration. Using the proxy configuration in Listing 1 as a template, add the settings in Listing 2.

Note that there are two caveats in the current nginx SSL implementation. The stable branch does not have certificate revocation list support. This issue has been addressed in unstable versions ≥0.8.7. Next, if you have a chain certificate file (sometimes called an intermediate certificate), you don't specify it separately as you do in httpd. Instead you need to add the information from the chain certificate to the end of your main certificate file. Do this by typing *cat chain.crt >> server.crt* on the command line. Once that is done, you won't use the chain certificate file for anything else; you simply point *ssl_certificate* to the main certificate file.

Static Content and Caching

With this basic setup working, the next step is for nginx to statically serve some images. This step will allow you to tune your back-end httpd processes for dynamic content serving. I'll also serve images with an expires header of 30 days,

Listing 1: Basic *nginx.conf*

```
user                          nobody;
worker_processes              2;

events {
       worker_connections   1024;
       use epoll;
}

http {
       include          mime.types;
       default_type    application/octet-stream;
       log_format      custom
                            '$http_host $remote_addr - $remote_user [$time_local]
                             "$request" '
                            '$status $body_bytes_sent "$http_referer" '
                            '"$http_user_agent"';
       access_log      /path/to/access.log custom;
       sendfile         on;
       server_tokens    off;

       upstream cluster {
           server 192.168.1.2 weight=1; // the weight can be adjust to send more
           server 192.168.1.3 weight=1; // traffic to specific machine(s).
       }

       server {
              listen 10.0.0.1:80;
              server_name www.domain.com domain.com;
              location       / {
                     proxy_pass            http://cluster;
                     proxy_redirect         off;
                     proxy_set_header       Host $host;
                     proxy_set_header       X-Real-IP $remote_addr;
                     proxy_set_header       X-Forwarded-For
                                               $proxy_add_x_forwarded_for;
                     proxy_buffers          8 32k;
              }
       }
}
```

Listing 2: SSL in *nginx.conf*

```
server {
        listen                  10.0.0.1:443;
        server_name             www.domain.com;
        add_header              Front-End-Https on;
        keepalive_timeout       70;
        ssl                     on;
        ssl_certificate         /path/to/server.crt;
        ssl_certificate_key     /path/to/server.key;

        location        / {
            proxy_pass              http://cluster;
            proxy_redirect          off;
            proxy_set_header        Host $host;
            proxy_set_header        X-Real-IP $remote_addr;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_buffers           4 32k;
            proxy_set_header        X-Forwarded-Proto https;
        }
}
```

which will cut down on the number of requests a client needs to make for common images that rarely change. To accomplish this, add the following to your server context:

```
location ~* ^.+\.(jpg|jpeg|gif|png)$ {
        root            /path/to/www;
        expires         30d;
}
```

If you'd like to disable logging for images requests, add the following line to the configuration:

```
access_log      off;
```

Listing 3: *gzip* in *nginx.conf*

```
gzip                    on;
gzip_http_version       1.0;
gzip_vary               on;
gzip_min_length 1100;
gzip_buffers            16 8k;
gzip_comp_level 5;
gzip_proxied            any;
gzip_types      text/plain text/css application/javascript text/javascript text/xml application/x-javascript;
gzip_disable            "MSIE [1-6]\.";
```

Next, nginx will gzip some output received from the httpd back-ends before sending it to the browser. The nginx server will only gzip certain content based on mime type and will completely disable gzip for some known-broken browsers. Add the code in Listing 3 to your server context.

If you'd like to cache some of your dynamic content with nginx, you have two options; file based or memcached based. If you're considering using nginx to cache content, be careful how you cache content that differs based on whether a visitor is logged in or not. To enable the file-based cache, add the following to the http context in your configuration file:

Listing 4: *memcached nginx.conf*

```
server {
  location / {
    set $memcached_key $uri;
    memcached_pass      name:11211;
    default_type        text/html;
    error_page          404 @fallback;
  }

  location @fallback {
    proxy_pass cluster;
  }
}
```

```
proxy_cache_path /data/nginx/cache  levels=1:2 keys_zone=one:10m;
```

The *levels* parameter sets the number of subdirectories for the cache, and the key and filename are an md5 of the proxyied URL, resulting in filenames similar to */data/nginx/cache/c/29/b7f54b2d-f7773722d382f4809d65029c*.

With the cache path set in the http context, you can now setup your cache in the http, server, or location context. To cache all 200 and 302 responses for 30 minutes and all 404 responses for 5 minutes, add the following:

```
proxy_cache             one;
proxy_cache_valid        200 302 30m;
proxy_cache_valid        404           5m;
```

If you'd prefer to use memcached for your cache, it's almost as easy (see Listing 4).

Server Statistics

Many monitoring systems support the *httpd mod_status* module to gather and trend statistics. The *stub_status* module serves a similar role with nginx. This module is not compiled by default and must be enabled with the *--with-http_stub_status_module* configure argument. Once the module is compiled in, add the code in Listing 5 to your configuration file. An HTTP request to *domain.com/nginx_status* will return a plain text response in the format shown in Listing 6.

Additional Modules

The *httpd mod_rewrite* module is used by many sites. While nginx does have a rewrite module, its syntax is slightly different from the one for httpd. The nginx wiki has the full details [4].

One example of rewrite feature is a simple rewrite to enable SEO-friendly member pages:

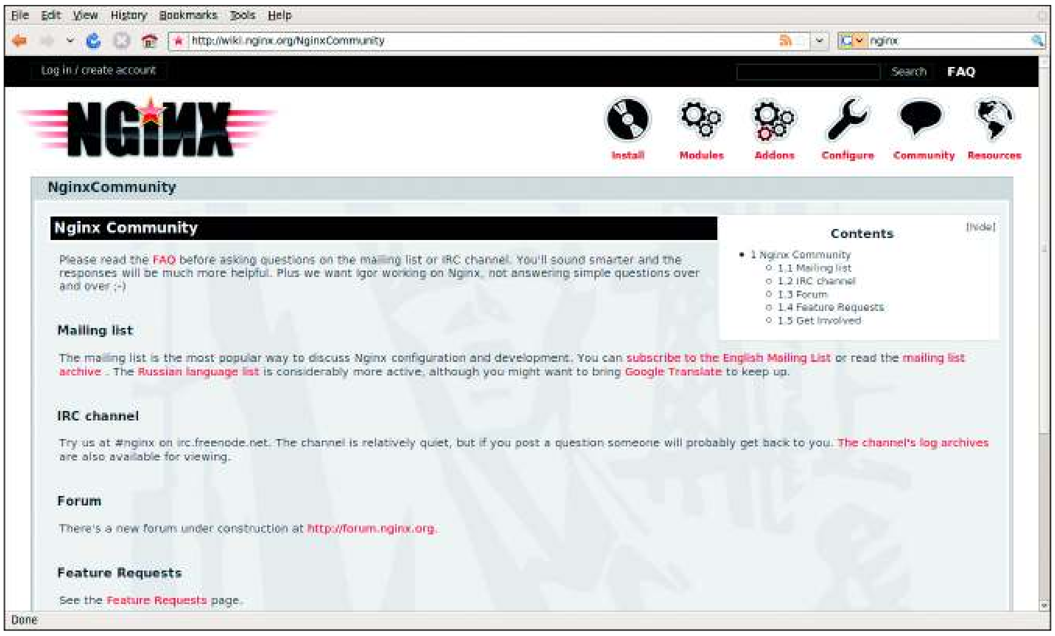
Listing 5: *stub_status nginx.conf*

```
location /nginx_status {
        stub_status             on;
        access_log              off;
        allow TRUSTED.IP.ADDRESSES
        deny                    all;
}
```

```
rewrite  ^/users/(.*)$ ➘
/user.php?user=$1? ➘
 last;
```

A more complicated rewrite uses an *if* condition to redirect your visitors to a consistent domain:

```
if ($host ~* www\. ➘
(.*)) {
  set $host_without_ ➘
www $1;
  rewrite ^(.*)$ ➘
  http://$host_ ➘
without_www$1 ➘
  permanent;
}
```



File Edit View History Bookmarks Tools Help

http://wiki.nginx.org/NginxCommunity

Log in / create account | Search | FAQ

Install | Modules | Addons | Configure | Community | Resources

NginxCommunity

Nginx Community

Please read the FAQ before asking questions on the mailing list or IRC channel. You'll sound smarter and the responses will be much more helpful. Plus we want Igor working on Nginx, not answering simple questions over and over ;-)

Contents [hide]

- 1 Nginx Community
  - 1.1 Mailing list
  - 1.2 IRC channel
  - 1.3 Forum
  - 1.4 Feature Requests
  - 1.5 Get Involved

Mailing list

The mailing list is the most popular way to discuss Nginx configuration and development. You can subscribe to the English Mailing List or read the mailing list archive . The Russian language list is considerably more active, although you might want to bring Google Translate to keep up.

IRC channel

Try us at #nginx on irc.freenode.net. The channel is relatively quiet, but if you post a question someone will probably get back to you. The channel's log archives are also available for viewing.

Forum

There's a new forum under construction at http://forum.nginx.org.

Feature Requests

See the Feature Requests page.

Done

Figure 1: In addition to configuration examples and resource links, the Nginx community provides a a mailing list, forum, and IRC channel.

The *GeoIP* module creates variables based on the IP address of the client matched against the *MaxMind GeoIP* binary files. The nginx *GeoIP* module has two prerequisites – *libGeoIP* and the *MaxMind GeoIP* database(s). The latest *libGeoIP* is available from the MaxMind site [5], but keep in mind that many distributions have *libGeoIP* in their package repositories.

Add the following two lines to your http context to enable the GeoIP module.

```
geoip_country  GeoIP.dat;
geoip_city     GeoLiteCity.dat;
```

You will now have the variables listed on *http://wiki.nginx.org/NginxHttpGeoIPModule* at your disposal. One common use case for the GeoIP module is to use the *$geoip_country_code* variable to send requests to different proxy upstreams based on country. If you'd like to pass the GeoIP information to you httpd back-ends, add the following to your proxy configuration:

```
proxy_set_header   HTTP_GEO   $geo;
```

Table 2 shows some additional nginx modules, along with a brief overview of their functionality.

Conclusion

Adding nginx as a caching reverse proxy to an existing httpd setup can significantly increase the performance of your existing infrastructure. Additionally, using some of the more advanced features in nginx will give you greater flexibility and might allow you to accomplish tasks that weren't feasible with your previous setup. I suggest you read the online documentation (Figure 1) and familiarize yourself with nginx before deploying it in a production environment. ■

Table 2: Nginx Modules

Module	Description
HTTP Referer	Filter requests based on the Referer header.
HTTP Limit Zone	Limit simultaneous connections from a client.
HTTP Limit Requests	Limit frequency of connections from a client.
User ID	Issue identifying cookies.
HTTP Addition	Append arbitrary text to pages.
FLV	Flash Streaming Video
Perl	Execute Perl directly within Nginx and call Perl via SSI.
WebDAV	WebDAV pass-through support.
Substitution	Replace text in pages.
Image Filter	Transform images with LibGD.
Secure Link	Protect pages with a secret key.
XSLT	Post-process pages with XSLT.

Listing 6: nginx_status output

```
Active connections: 291
server accepts handled requests
 16630948 16630948 31070465
Reading: 6 Writing:
179 Waiting: 106
This server has 291 active
connections, has accepted
and handled 16630948 connections
while serving 31070465 requests...
```

INFO

[1] nginx: *http://www.nginx.org*

[2] nginx Modules page: *http://wiki.nginx.org/NginxModules*

[3] mod_rpaf: *http://stderr.net/apache/rpaf/*

[4] Nginx Rewrite module: *http://wiki.nginx.org/NginxHttpRewriteModule*

[5] MaxMind: *http://geolite.maxmind.com/download/geoip/database/*

THE AUTHOR

Jeremy Garcia is the founder and admin of LinuxQuestions.org, which uses nginx to reverse proxy all of its content and is one of the largest Linux communities on the web. He has been using Linux for more than 10 years and is an ardent but realistic Open Source evangelist.