Nginx notes

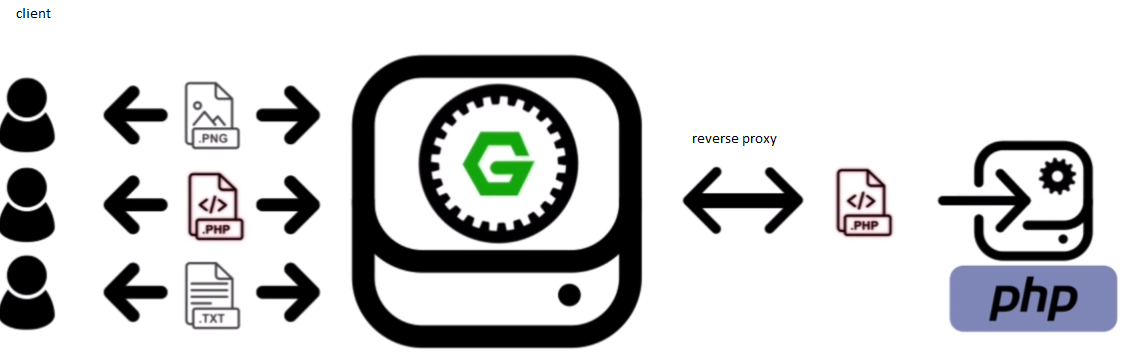
# Configuration

About Nginx(Igor Sysov 10k concurrent conections, high performance, high concurrency, low resource usage). It is web server but in its core it is a reverse proxy server.

Nginx vs apache:

Basic architecture of apache is prefork mode (had spawned a set number of processors, each of which can serve a single request at a time regardless wheathe that request is for php script of image)

Nginx is dealing with requests asynchronously (single process can serve multiple requests concurrently that number depends on system resource available to nginx process) –sidenote unlike apache can’t embed server side programming languages into its own processes meaning that all requests for Dynamic content has to be dealt with completely separate process like php fpm and then reverse proxy to the client via enginx (low resource usage because not hav9ing to deal directly with embedded programming languages). But unlike apache, server side language modules don’t need to be run for every single request that server receives, instead nginx will handle serving static resources without php ever knowing about it(apache will handle every request with that costly overhead)-that is saving on system resources



Nginx is faster than apache (internet collection will allow speed of getting data to the client, but can serve static resources much faster (no need for any server side languages), and handle a much larger number of concurrent requests (can receive 1k of requests on single processing thread. Apache will accept request up to preconfigured number and reject rest) ).

Nginx configuration (requests are interpreted as uri location \*location /images {}\* , this allows nginx to work not only as web server, but also as load balancer to mail server, whereas apache default to filesystem locations \*<Directory “/www/site/images”>\*). .htaccess file can override configuration, there is no such in nginx.

# Install Nginx

* Using package manager

Ssh [anel@163.172.171.154](mailto:anel@163.172.171.154) (to remove ip/remove it from .*ssh/known*\_hosts: *ssh-keygen –R 163.172.171.154* )

With package manager: quick and easy, limited install options, no support for additional modules. Suitable for most basic web servers or testing and development. Almost never best solution

**sudo apt-get install nginx**

With this particular package nginx is also running ( search for process command for all users listing detailed information and include boot processes*=> ps aux | grep nginx* => should have master and worker processes). Ifconfig = > ip in the browser to see does nginx works and listens oh http port 80.

Check for nginx configuration files : **ls –l /etc/nginx**

* Building from the source

<http://nginx.org/>

<https://www.nginx.com/> [nginx+ paid]

<https://www.nginx.com/resources/wiki/>

Main benefit why to build from the source is to have ability to add custom modules or to extend standard nginx functionality (which you cannot do with package manager). Nginx modules exists in 2 form: bundled modules(gzip,spdy,ssl,geoip – come with the source), third party modules.

Navigate to download link: <http://nginx.org/en/download.html>

We will use mainline version.

Copy link: => <http://nginx.org/download/nginx-1.15.5.tar.gz> and use wget. Navigate to workspace folder to get tarball format:

**$wget** [**http://nginx.org/download/nginx-1.15.5.tar.gz**](http://nginx.org/download/nginx-1.15.5.tar.gz)

**$ tar –xzvf nginx-1.15.5.tar.gz**

**$cd nginx-1.15.5**

Configure source code for the build.

To do this run the configure script in the source code directory. Before enable tool to compile nginx (gcc), and some dependencies : libpcre3, libpcre3-dev, zlib1g zlib1g-dev (for g zipping), libssl-dev (for ssl/https support)

**$ apt-get install libpcre3 libpcre3-dev zlib1g zlib1g-dev libssl-dev**

**$ apt-get install build-essential**

**$./configure**

We still haven’t added any custom configuration flags. To do so run **./configure –help** to see all of them. See documentation <http://nginx.org/en/docs/configure.html>

|  |  |
| --- | --- |
| **Flag** | **Description** |
| **--sbin-path** | = /usr/bin/nginx  location of nginx executable which will use to start and stop the nginx service (usr/bin common location for Ubuntu executables) |
| **--conf-path** | = /etc/nginx/nginx.conf  Path of nginx configuration files. |
| **--error-log-path** | = /var/log/nginx/error.log |
| **--http-log-path** | = /var/log/nginx/access.log |
| **--pid-path** | =/var/run/nginx.pid  We will need to know when configuring nginx |
| **--with-pcre** | Tell the nginx to use system specif library for regular expressions |
| **--with-http\_ssl\_module** | Bundled module |

**$ ./configure --sbin-path=/usr/bin/nginx --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --with-pcre --with-http\_ssl\_module**

We can now go ahead and compile the configuration source by running make:

**$ make**

After this is done install the compiled source with make install:

**$ sudo make install**

See the configuration files: $ ls –l /etc/nginx ; See the binary : **$ls –l /usr/bin |grep nginx**.

Test the version: **$ nginx –V** (see the arguments which should be same as configured). Test **$ps aux|grep nginx** and run from source folder nginx: **$ sudo nginx**

Navigate to the browser to check again.

|  |
| --- |
| *Configuration summary*  *+ using system PCRE library*  *+ OpenSSL library is not used*  *+ using system zlib library*  *nginx path prefix: "/usr/local/nginx"*  *nginx binary file: "/usr/local/nginx/sbin/nginx"*  *nginx modules path: "/usr/local/nginx/modules"*  *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"* |

## Configuring the system service for nginx (as systemd):

Adding nginx as systemd (newer standard for services, >Ubuntu 15.0.4) service (will enable us to start/stop/restart, reload(configuration) and start on boot.

To see options for nginx navigate to source folder and run : **$nginx –h**

To stop the service use signal flag (-s) and stop (terminated) : **$nginx –s stop**

* **Let’s add systemd service:**

To enable the service we are going to have to add a small script which being the same across all operating systems (init.d are old)

Nginx initscripts : <https://www.nginx.com/resources/wiki/start/topics/examples/initscripts/>

$ touch /lib/systemd/system/nginx.service

This file already exist in /lib/system/system folder but it is different (from Feb 11, 2017)

|  |
| --- |
| **[Unit]**  Description=The NGINX HTTP and reverse proxy server  After=syslog.target network.target remote-fs.target nss-lookup.target  **[Service]**  Type=forking  PIDFile=/var/run/nginx.pid  ExecStartPre=/usr/bin/nginx -t  ExecStart=/usr/bin/nginx  ExecReload=/usr/bin/nginx -s reload  ExecStop=/bin/kill -s QUIT $MAINPID  PrivateTmp=true  **[Install]**  WantedBy=multi-user.target |

Start with ngingx (needed to do daemon reload) :

*Warning: nginx.service changed on disk. Run 'systemctl daemon-reload' to reload units.*

**$ sudo systemctl daemon-reload**

**$sudo systemctl start nginx**

With systemd service enabled, we can also check nginx status using systemd instead of process command. **$sudo systemctl status nginx**

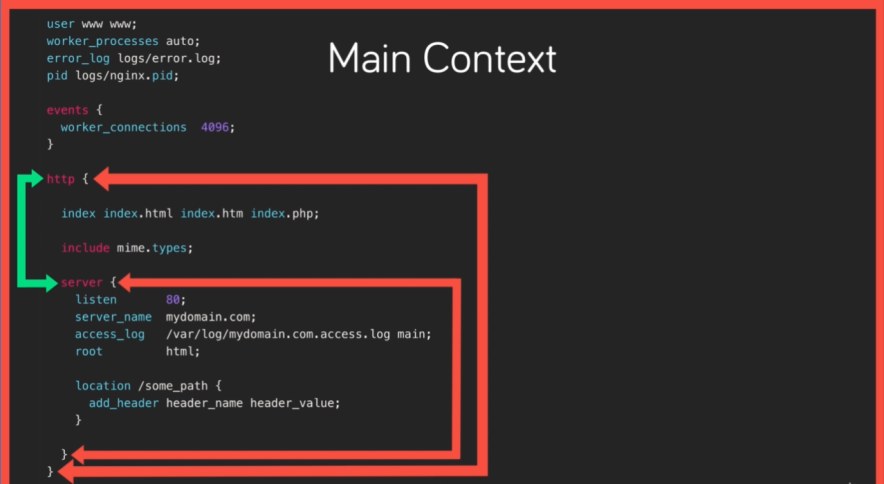
Enable startup on boot : **$sudo systemctl enable nginx**

|  |
| --- |
| *Synchronizing state of nginx.service with SysV init with /lib/systemd/systemd-sysv-install...*  *Executing /lib/systemd/systemd-sysv-install enable nginx* |

## Configuration terms:

*Context –* section within the configuration, where directives can be set for that given context. It is like scope (::) and like scope they can be nested and inherit from parents.Topmost context is configuration file itself - Main context (global directives that apply to master process). Http context , server context(apache V host), location context(matching uri locations on incoming requests to the parent server context)

*Directive* – specific configuration option that get set in configuration file (consists of name and value : *server\_name mydomain.com*)



## Creating a Virtual Host or server context:

To serve static files from directory on our server.

$ *mkdir my\_sites && cd my\_sites && mkdir demo && cd demo*

Server context is responsible for listening on some port (80 http, 443 https) for given ip address or domain.

Start with */etc/nginx/nginx.conf* file : add server context and *directive listen 80; server\_name IPadess; root /home/anel/..* and for variety of types in http context *include mimo.types*;

Using *nginx –t* you can verify does you configuration is successful.

|  |
| --- |
| *nginx: the configuration file /etc/nginx/nginx.conf syntax is ok*  *nginx: configuration file /etc/nginx/nginx.conf test is successful* |

Using *systemctl reload nginx* you can load a new configuration (with *systemctol restart nginx = >* it will stop and if it is not good old configuration will not be reloaded). Mime.types are used for providing the server content type for various of extensions.

To test the MIME type check the stylesheet header using curl (content-type: text/plain is not good) ; **curl –I http://163.172.171.154/style.css**

## Location context:

Added in server context.

According to the priority we have:

* Exact match :*location = /anel {return 200 “Status message!”;}* # when returning return status code and message
* Preferential prefix match : *location ^~ /Anel7 {return 200 “Status message!”;}*
* Regex match (case sensitive, for case insensitive put *~\**) : *location ~ /anel[0-9] {return 200 “Status message!”;}*
* Prefix match (can be treated as /anelh, /anelax/mariadb etc) *location /anel {return 200 “Status message!”;}*

## Variables:

Aside of concept of scope, includes there are variables and conditionals in nginx.

There are 2 types of variables:

* Configuration variables (user defined, we set ourselves used with **set** keyword (set $anel ‘Anel’))
* Nginx builtin global/modules variables (Alpabetical index of variables with modules that makes it available <http://nginx.org/en/docs/varindex.html> ).Modules like ngx\_http\_core\_module, ngx\_http\_log\_module are already part of nginx (not needed to add them manually).

***$host, $uri,***

***$args*** (test it with <http://163.172.171.154/demo?name=anel> ) will return *name=anel.*

To obtain only anel from query try with***$arg\_name****!(based on the query string nginx compiles named variable for each parameter prefixed with arg),*

***$date\_local*** *(*holds ISO date time string for the local time *:* Sunday, 07-Oct-2018 13:52:59 UTC*)*

*Basic conditionals*

Use of nginx conditionals inside nginx location context is highly discouraged (<https://www.nginx.com/resources/wiki/start/topics/depth/ifisevil/> ) (use only **return** and **rewrite** inside location context, everything else could give **SIGSEGV**).

When using “**if ($arg\_name != “anel”)**” make space between if and ( , as well as between arguments and operator hence it will be an error.

## Rewrites and redirects:

**rewrite pattern uri**

**return status\_code message**

In case where status\_code being 307 variant (which is for redirects) return behavior changes it accept URI as a second parameter.

But when using : return 307 /thumb.png url will be redirected, but the url wil be changed from route from location context to redirected link (/thumb.png), what is a main difference between rewrite and redirects ! Redirect tells the client preforming the request where to go instead, while rewrite mutates request internaly.

Rewrite: In server context before matching the request to any location context, we should write : **rewrite ^/user/\w+ /greet** (meaning first argument: regex **^** ==starting with, **\w+** more than 1 word character) (meaning second argument: new url to rewrite). The browser will be pointing to the same location of first argument!

Note: When uri is rewritten it is also re-evaluated by nginx, as a completely new request !!! Meaning that new /greet will start from the top of nginx.conf and will get re-evaluated again ! Reevaluation makes rewrites more powerful but requires more system resources then return.

Feature of rewrite is ability to capture certain parts of the original url using standard regex capture groups (wrap that part in braces, and the capture group can be accessed with $). For capturing user name:

**rewrite ^/user/(\w+) /greet/$1**

So now /user/anel will be rewritten to /greet/anel. After /greet/anel will be reevaluated , skipping past rewrite and exact match will take priority instead (if exist) prefix match.

Passing the **last** flag:

Add new rewrite bellow the previous ; **rewrite ^/greet/anel /thumb.png.** First rewrite will match to the same route, it will be rewritten to /greet/anel and will be re-evaluted and will match the second rewrite and again will be rewritten as /thumb.png and served as such.

With last flag will tell url not to be rewritten any more. So in this case with modification of the first rewrite as: **rewrite ^/user/(\w+) /greet/$1** l**ast**

/user/anel will be rewritten to /greet/anel and after re-evaluation it will skip the second rewrite.

**rewrite ^/user/(\w+) /greet/$1**

**Try files & named locations**:

Can be used in server context :

**try\_files path1 path2 final;** applying to all incoming request, or in location context .Have nginx check for resource to respond worth in any number of locations relative to the root directory with a final argument that result in a rewrite and re-evaluation as with rewrite directive.

**try\_files /thumb.png /greet; (**thumb relative to the root directory does exist and will be served (even if url doesn’t exist it will be served !) if doesn’t exist it will be skipped and tested with /greet).

Note: when try files reach last argument, that file is treated as internal rewrite, meaning that rewritten request will also be re-evaluated and caught by location !

Named locations: assign name to location context, and use directive such as try\_files use that location by its name, ensuring no re-evalution has to happen to the final argument, but instead definite call to that location. Use at sign @

## Logging:

* Error logs (for anything that failed and didn’t happen as expected)
* access logs( to log all request to the server)

<http://nginx.org/en/docs/ngx_core_module.html#error_log>

<http://nginx.org/en/docs/http/ngx_http_log_module.html>

<https://docs.nginx.com/nginx/admin-guide/monitoring/logging/>

Enabled by default !

During the installation we have set the path to the error and access logs (/var/log/nginx/error(access).log). Permission to view log files is granted to users being in **adm** group (created by nobody, owned by adm):

*$sudo usermod –aG adm anel*

Clear those files: $ sudo *echo ‘’>access.log && echo ‘’>error.log (Permission denied)*

*404 response can be viewed from access log and is valid response,but sometimes if url-ed to something that doesn’t exist will be shown in error.log. To simulate failing of nginx add second parameter to the root directive in .conf file reload nginx and see the log.*

*While having global access log, we can define location specific access logs, which are created by the running nginx : define in location context* ***access\_log /path/to/{custom\_name}.access.log*** *With such directive logs are redirected to this path instead to global access log. By adding another access log with the global path we will have both access logs!*

*Disabling for certain directives:* ***access\_log off*** *Doing so we are reducing server load and keeping log files smaller, userful for site receiving high traffic.*

*Check the last line of error log : $ tail –n 1 /var/log/nginx/error.log*

For more advanced logging configuration we can specify *format,* enable *gziping* of the log file*,* or set the log *level* for error logs.

## Ineritance and directive types:

Nginx context (for example loation context or scope) inherits configuration from its parent context.

Directive types:

* Standard directive – most common, and can be declared once in a given context. Root directive.Inheritance same as array = parent to child with child being able to override with its own declaration.
* Array directive – can be declared multiple time without overriding the previous. We can have access\_log directive declared in *main* context which will be inherited in *http*, *server* context,but if in server context we disabled **access\_log** it will be inherited for childs (locations) of server context except they changed their **access\_log** directive.
* Action directive – invoke some or break action in configuration (redirect via **return** directive or rewrite via **rewrite** or try\_**files** directive)

## php processing:

Up to now we have dealt with configuring nginx to serve static files of various types leaving the rendering of that file be handled by client based on its content type or mime type.

Ability to serve dynamic content that is generated from server side language such as php, nginx isn’t able to embed its server side language processors. Instead we will configure standalone php service “php-fpm” to which nginx will pass the request for processing and then upon receiving the response from php-fpm typically as html return that to client. This is essentially nginx functioning as reverse proxy server. Nginx will communicate with php-fpm through FastCGI protocol (protocol like http for transferring binary data: <https://www.digitalocean.com/community/tutorials/understanding-and-implementing-fastcgi-proxying-in-nginx>).

Install php-fpm service using package manager: apt-get update && apt-get install php-fpm

It will install 8 new packages to apt (php-common php-fpm php7.0-cli php7.0-common php7.0-fpm php7.0-json php7.0-opcache php7.0-readline) (should be 7.1 ??).

Already configured as systemd service. To be sure list all systemd services:

***$ sudo systemctl list-units | grep php***

*php7.0-fpm.service loaded active running The PHP 7.0 FastCGI Process Manager*

Check the status of new service, which is already running (single master process and 2 workers):

**$ sudo systemctl status php7.0-fpm.service**

|  |
| --- |
| php7.0-fpm.service - The PHP 7.0 FastCGI Process Manager  Loaded: loaded (/lib/systemd/system/php7.0-fpm.service; enabled; vendor prese  Active: active (running) since Mon 2018-10-08 23:13:38 UTC; 4min 58s ago  Main PID: 25348 (php-fpm7.0)  Status: "Processes active: 0, idle: 2, Requests: 0, slow: 0, Traffic: 0req/se  CGroup: /system.slice/php7.0-fpm.service  ├─25348 php-fpm: master process (/etc/php/7.0/fpm/php-fpm.conf)  ├─25351 php-fpm: pool www  └─25352 php-fpm: pool www  Oct 08 23:13:38 scw-95fca6 systemd[1]: Starting The PHP 7.0 FastCGI Process Mana  Oct 08 23:13:38 scw-95fca6 systemd[1]: Started The PHP 7.0 FastCGI Process Manag |

Now we need to pass incoming requests to the service ! Lets do it in nginx.conf file.

First add in *server* context *index* directive (which file to load if request point to the directory, current value is *index.html; typically we want index.php and if not exist then index.html (like try\_files)=> index index.php index.html*).

**Location /** (try\_files $uri $uri/ =404) (prefix match for anything, request uri, requested uri for matching the directory with php or html, and finaly default nginx 404)=>any request with static content.

**Location ~\.php (**anything ending with php in order to pass it to php-fpm which will do by means of fastCGI protocol, have priority because of regex**).** Beforepassing request to php-fpm from location context, we have to include some paraemters: **include fastcgi.conf** (located in /etc/nginx/), need to pass unix socket created by php-fpm (socket->http port, a file on wich a service can listen for binary data; find that socket with: **find / -name \*fpm.sock** => (*/run/php/php7.0-fpm.sock*)) with **fastcgi\_pass** **unix:****/run/php/php7.0-fpm.sock**

**Create a php file: $ echo ‘?php phpinfo(); ?’ > ‘/home/anel/eacon/django/nginx\_workspace/nginx-1.15.5/mysites/demo/info.php’**

When trying to connect 502 (bad gateway). To check => With **ps aux | grep nginx** => default worker process is nobody

|  |
| --- |
| root 25758 0.0 0.0 6568 1844 ? Ss 23:43 0:00 nginx: master process /usr/bin/nginx  nobody 25759 0.0 0.0 6740 2476 ? S 23:43 0:00 nginx: worker process |

With **ps aux | grep php** => default worker process is www-data.

|  |
| --- |
| root 25348 0.0 0.2 118988 23772 ? Ss 23:13 0:00 php-fpm: master process (/etc/php/7.0/fpm/php-fpm.conf)  www-data 25351 0.0 0.0 118988 5968 ? S 23:13 0:00 php-fpm: pool www  www-data 25352 0.0 0.0 118988 5968 ? S 23:13 0:00 php-fpm: pool www |

**Conclusion:** nobody user doesn’t have permission to php-fpm socket (change permission of the socket, but secure is to configure nginx to run as the same user (www-data)!) In nginx.conf in main context add: **user www-data**

Processing index.html : $ echo ‘<h1> Date: <?php echo date(“l jS F”); ?> </h1>’ > **‘/home/anel/eacon/django/nginx\_workspace/nginx-1.15.5/mysites/demo/index.php**

## Worker processes:

* Master process – actual nginx service (/usr/bin/nginx)
* Worker process – master process spawns workers processeses which listen for and responds to client requests. Default number for worker processes is 1 (use **worker\_processes** directive in the main context). They are asyncrhonouse meaning they will handle requests as fast as hardware is capable of and creating the second worker process doesn’t increase hardware ability.Multicore processor cannot share processes ,meaning single nginx processes (worker) can be run on a single cpu core (*nproc, or lscpu*)=> number of cores=number of workers. Instead of setting it to 8 we can set worker\_processes to auto and have 1 worker per each cpu core!
* Inside *events* context we can set directive **worker\_connections** to number of connects each worker process can accept (how many files can be opened at once for each cpu core- to check *open file limit* run **ulimit -n**(1024)) =>worker\_processes x worker\_connections = max connections of concurrent requests our server should be able to accept.
* Pid directive - added during configuration file in /var/run/nginx.pid (reconfigure process id location via configuration file) **pid /var/run/new\_nginx.pid** in main context of nginx.conf file.

## Buffers and timeouts:

These parameters are dependent on the request to the server. Leave these values default.

Buffering in brief is when a process or nginx worker reads data into memory(ram) before writing it to its next destination. For example,nginx receives request which it reads from tcp port 80 , writes these request data to memory which is buffering , or if buffer is too small for the amount of data being read, write some of it to disk(buffer overflow).

In opposite, nginx response to a request with for example static file, which it reads from disk into memory (buffering the file) and send data to the client from memory.

Buffer – layer of protection between reading and writing of data.

Timeout – cut off time for a given event. If receiving a request from client stop after certain number of seconds, thus preventing the client of sending the endless stream of data and breaking the server.

For http context: **client\_body\_buffer\_size 10K;** # buffer size for POST submissions (most likely from form submission) k or K in bytes (<http://nginx.org/en/docs/syntax.html> ), **client\_max\_body\_size 8M;** meaning don’t accept post requests more than 8Mbytes (larger server will respond with 413 error => request entity too large). **Client\_header\_buffer\_size 1K;** (for reading request header it is enough).

Client timeouts :**client\_body\_timeout** and **client\_header\_timeout (**default **60s** body doesn’t refer to time to transmit the entire request body, but rather the time between consecutive read operations (that happen to the buffer))(number only (10) is 10 ms). **Keepalive\_timout** sets the amount of time nginx shoud keep a connection to a client open for in case more data is on the way (useful when client is requesting a number of files and keeping a connection open reduces the time it takes to open another new connection, this can influence on pool of max\_connections being used up –can be 15 or 1-2s).**send\_timeout** directive , if the client doesn’t receive any of response date in this amount of time, doesn’t have to be all of response data, but non of it at all,. **Sendfile on (**skip buffering for static files (when sending data to client from disk, don’t use a buffer…read that data from the disk and write it directly to the response), **)** and **tcp\_nopush on** (enable nginx to optimize the size of those data packets being sent to the client). The last two will help to optimize site with large number of stsatic resoureces.

## Adding Dynamic Modules:

Making more functionality available == adding new nginx modules.

Adding nginx modules to existing nginx install by rebuilding the source, but more specifically do so with dynamic modules

*Dynamic modules* = modules we can add selectively from nginx configuration unlike

*Static modules* = loaded before we start

Adding standard modules is same as adding dynamic modules, the same as with upgrading nginx.

Rebuild nginx from source: go to folder with source code, see configuration for the current installation (**nginx –V** or **/usr/bin/nginx -V**). Copy this configuration :

|  |
| --- |
| --sbin-path=/usr/bin/nginx --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --with-pcre --with-http\_ssl\_module |

To see all modules run within the same directory: **nginx –-help .** Some of them ends with **=dynamic (nginx --help | grep dynamic).**Example is --with-http\_image\_filter\_module=dynamic (<http://nginx.org/en/docs/http/ngx_http_image_filter_module.html> )

|  |
| --- |
| **./configure** --sbin-path=/usr/bin/nginx --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --with-pcre --with-http\_ssl\_module --with-http\_image\_filter\_module=dynamic --modules-path=/etc/nginx/modules |

Configuration will fail, because dynamic module requires dependency GD libarary (image processing module for linux). (apt-get install libgd-dev). Then compile by **make** and **sudo make install** (which installs newly compiled source over existing install). Again check **usr/bin/nginx –V**.

To use this module go to nginx.conf and apply it to the .png file. Try to use inside some location when getting the thumb.png for example (**image\_filter rotate 90**) this will be unknown directive. Because it is dynamic it is not loaded by nginx by default, it exists in modules path as shared library (modules have common .so extension). To load it we need to add in *main* context **load\_module modules/nginx\_http\_image\_filter\_module.so**

I have a problem of having two instances of nginx installed so instead of using relative path to the root in nginx.conf I need to use absolute path so : **load\_module /etc/nginx/modules/nginx\_http\_image\_filter\_module.so**

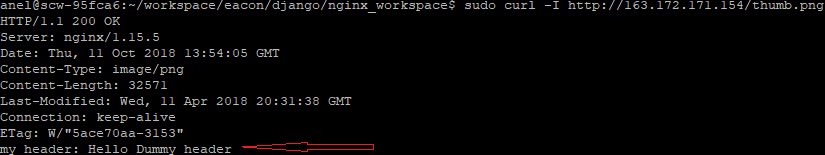
# Performance

## Headers and Expires

Expires headers are responses from server informing the client how long it can cached that response for. If we have a photo on website, that photo data isn’t going to change so often, meaning we can tell the browser to cache the copy of a photo for relatively long time and doing so avoid any future requests for that photo, to improve web site load times and often overlooked a reduction in requests to server. Css files can change more frequently, we can set it to shorter time, but still browser can cache it.

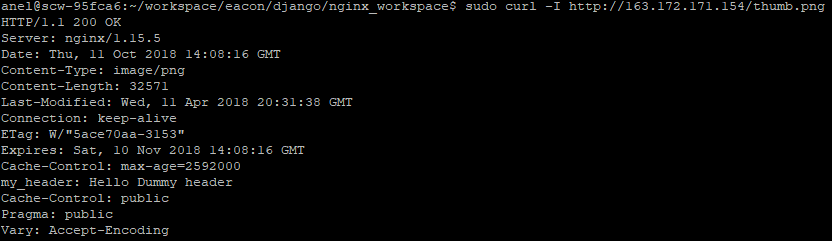
*Set generic response header*:

add\_header directive in location : **add\_header my\_header “Hello header”,** reload it and to test it make curl request to location where this header is added with *–I flag for headers*. *curl –I https://163.172.171.154/thumb.png* = >



So we can set this location as typical static resource by :

* Set Cache-Control header: **add\_header Cache-Control public;** #Telling the receiving client that this resource or response can be cached in any way.
* Set Pragma header: **add\_header Pragma public;** #Older version of Cache-Control header
* Set Vary header: **add\_header Vary Accept-Encoding;** #Meaning content of this response can vary based on the the value of the request header except encoding; will see later about value Accept-Encoding
* Set expires directive: **expires 1M;** #Expires directive sets expires header on this response ;we can set standard nginx duration 60m (minutes), 1M(month).



Cache-Control header is added as max-age=2592000 ( number of seconds in 30days). Using this, browser will cache thumb.png locally on users machine for this amount of time before requesting it from our server again.

Typical location for static resources might look like (with expires headers):

(case insensitive regex **~\***), looking any request ending with extension (.png,etc..)and use directive: **access\_log off** (unless we want to log each and every request!).

**location ~\* \.(css|js|jpg|png) $**

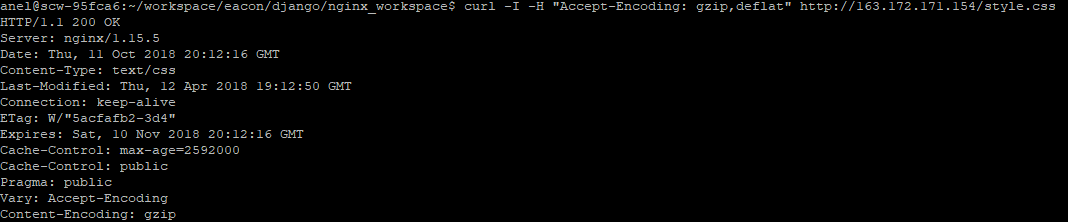
Now if we go to*curl –I* [*https://163.172.171.154/style.css*](https://163.172.171.154/style.css) we should see the same expires headerin order to avoid or minimize server side language processing

## Compressed Responses with gzip

Improved static resource delivery, by configuring compressed responses.

When a client request a resource (typically static file .css, .js), that client can indicate his ability to accept compressed data (Accept-Encoding:gzip). What are all modern browser capable. That means we can compress our response on the server typically using gzip which reduces the size of file and reduces the time for the client to receive that response, which client can be decompress before rendering.

* Enable gzip compression = > in *http* context (applying to all of our server and location blcoks, but any child can override it) use gzip directive (which comes with nginx core) **gzip on;**
* Configure enabled compression (level of compression, use higher number and more server resources, but will have as a result smaller files 3or 4) = > **gzip\_comp\_level 3;**
* Set mime types as arguments: **gzip\_types text/css text/javascipt**; (alternatively, because gzip\_types is array directive we could write it one bellow another)
* Before all client must indicate they are willing to accept compressed file. In add\_header must be with Vary argument (Vary header). Resposne can vary based on Accept-Encoding the client sends with request that variation being compressed or uncompressed. To do so set our own header on request with –H flag, as well as other compression formats such as deflat (zlib) so: ***curl –I –H “Accept-Encoding: gzip, deflat”*** [***http://163.172.171.154/style.css***](http://163.172.171.154/style.css) ***.****Without* ***–I*** *flag we got binary output (gzip).*



To test it Uncompressed: ***curl –I*** [***http://163.172.171.154/style.css***](http://163.172.171.154/style.css) ***> style.css,*** *Compressed****: curl –I –H “Accept-Encoding: gzip, deflat”*** [***http://163.172.171.154/style.css***](http://163.172.171.154/style.css) ***> style.min.css. ls –la style\**** *(size is reduced almost by double).*

## FastCGI Cache:

Nginx micro cache is simple server side cache that allows us to store dynamic language responses.

<http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html>

For websites relying on php and mysql this cache can provide incredible performance benefits and drastically reduce server load.

Path: user makes some request to nginx for dynamic content, nginx passes that to php\_fpm which processes and even include some database queries. Php\_fpm then returns that response to nginx most likely in form of html, at which point nginx can write that response using micro cache to disk, meaning next time a user makes same request there is no need to contact for php\_fpm.

How to setup micro cache for php using nginx fastcgi cache?

* To enable fastcgi caching making available to all server context write in http context :

**fastcgi\_cache\_path /tmp/nginx\_cache levels=1:2 keys\_zone=MYCACHE:100m inactive=60m;**

**fastcgi\_cache\_path** **fastcgi\_cache\_path** **fastcgi\_cache\_path** directive takes file system location to which cache entries can be written (use **/tmp/** which gets emptied on boot for most linux distro). **Levels** parameter – allows us to configure the depth of directories to split cache entries in 2. Without this, nginx will add every cache entry in the directory in path ,which is ok for smaller sites. With this there will be directory which ends with the last character in entry name and second directory which has 2 character in name and there are last 2 characters in hash of cache entry name.**Key\_zone** use any descriptive name (MYCACHE, or ZONE\_1) : SIZE OF cache zone (100MB,. 100m). **Inactive** parameter says how long to keep cache entries after the last time it’s accessed (10minutes is default, if ommited)

* **fastcgi\_cache\_key** directiveis a standard directive which takes a string format from which to create cache entries. The string is hashed (md5).

**fastcgi\_cache\_key “$schema$request\_method$host$request\_uri”**

$schema = > http/https ; Without this it will work, but with this we will **have 2 entries.**

$request\_method = > get, post, etc

$host => domain

$request\_uri. Typical cache entry will be identified as for example : https:// GET domain.com /blog/something

* We have to implement where we want dynamic content to be cached. In some location, where to enable caching, we have to reference cache name(identifier): **fastcgi\_cache\_ MYCACHE**
* To fine tune validity of cache responses and specify how long a cache response should be valid for based on response code 200, 404 for 60 minutes**: fastcgi\_cache\_valid 200 60m; fastcgi\_cache\_valid 404 10m;**

**Test current performance without micro cache**: simplest tool is apache bench (http benchmarking) **: $ apt-get install apache2-utils**

**Test the tool: $ ab**

**$ curl** [**http://163.172.171.154/**](http://163.172.171.154/)

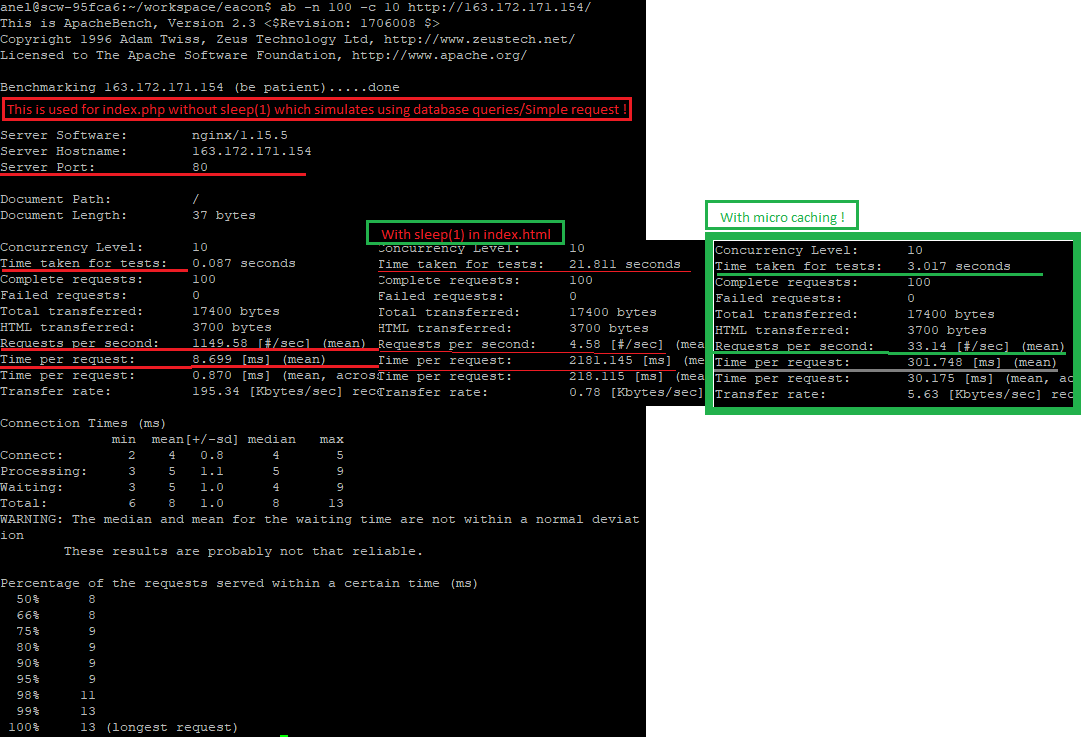
To test it create 100 requests to the page in lots of 10 connections = > 10 connections at a time 10 times =>100 requests

Using apache bench => -n (number of requests) , -c with concurrency of 10 connections.

<https://httpd.apache.org/docs/2.4/programs/ab.html>

**$ ab –n 100 –c 10** [**http://163.172.171.154/**](http://163.172.171.154/)

So nginx will cache 1. Request and cache 99 almost immediately.



\* Fastcgi module allows us to check the status of the response meaning whether is served as cache or not by means of variable **($upstream\_cache\_status** , which we can pass with response as a header – add that to all responses to http context. X is a naming convention for custom headers) **add\_header X-Cache $upstream\_cache\_status.** Reload configuration, do a curl request for the headers (-I) => X-Cache: **HIT** (this was served from the cache).

\* By making url explicit /index.php => X-Cache:**MISS** (it is not cached, but it will be a new cache entry, next time when calling will be HIT).

\* Adding cache exceptions (for locked area of site,life data or etc): First create a variable with which to track cache exceptions in server context (default no exceptions **set $no\_cache 0**;). Add condition to change this variable ($arg\_skipcache = 1) an dthan in location context add **fastcgi\_cache\_bypass $no\_cache;** and **fastcgi\_no\_cache $no\_cache;** (if no\_cache is 1 bypass serving from the cache and don’t write the response to the cache. Test it with curl –I **/?skipcache=1**  => X-Cache: **BYPASS**

## HTTP2:

<https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-in-ubuntu-16-04>

From nginx 1.9.5, nginx includes http2.

* Binary protocol (more compact way of transferring data using 0,1, less errors during data transfer) , http1 is textual protocol.
* http2 Compresses responses headers which reduces transfer time
* http2 uses persistent connections which are multiplexed
* Multiplex streaming (multiple assets css,js,html can be combined in a single stream of binary data and transmitted over a single connection). Http1 on other hand requires dedicated connection for each resource.
* Can perform server push (client can be informed if assets(css,jv,index)go along with initial request of page)

Opening a new connection is a time consuming process (concatenate multiple js, css into single file).

Require handshake between client and server and for this to happen headers need to be passed on both ends each time. There is a limit how many concurrent connection the browser can open to a particular domain at once.

For the basic page with html, js and css:

\* Http1 uses simplex streaming (1 connection handles 1 request) => Client request index.html, which opens a new connection to the server. Server respond with html content (connection 1). Browser has to process html data from which it can identify linked resources required by the html page (style.css and script.js).Again client opens new connection and requires style.css to which server response with css data (connection 2). Same for script.js (3 connections).

\* Http2 uses multiplex streaming: Client request index.html, which opens a new connection to the server. Server sends html data and browser processes data and over the same persistent connection requests for .css and .js files to which a server sends a multiplex stream containing the data for both files. Only 1 connection !

To enable http2 => Requirement is SSL or HTTPS (it is already added in configuration of nginx installation)

* Add http2 to our install: cd nginx-1.15/ && nginx –V (copy )
* To see the flag needed = > **./configure –help | grep http\_v2 => --with-http\_v2\_module**

|  |
| --- |
| ./configure --sbin-path=/usr/bin/nginx --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --with-pcre **--with-http\_ssl\_module** --with-http\_image\_filter\_module=dynamic --modules-path=/etc/nginx/modules **--with-http\_v2\_module** |

* Make
* Sudo make install
* Sudo systemctl restart nginx
* Sudo systemctl status nginx

To enable SSL we will need some certificate (self-signed and private key). For production website we will need a legitimate certificate from a avendor or a service such as *let’s encrypt.*

***$ openssl req –x509 –days 10 –nodes –newkey rsa:2048 –keyout /etc/nginx/ssl\_anel/self.key –out /etc/nginx/ssl\_anel/self.crt***

*Lets enable SSL : in nginx.conf =>* **listen 80** to **listen 443 ssl (**standard http port for ssl encrypted connections, add **ssl** torefer that ssl module being used for listen directive **).** Add ssl certicate and tell nginx where to find the signing key:

**ssl\_certificate /etc/nginx/ssl\_anel/self.crt;**

**ssl\_certificate\_key /etc/nginx/ssl\_anel/self.key;**

Reload!

Use <https://163.172.171.154> ( on Mozila add security exception and it will be a lock ! On Chrome security will be strikethrough for https).

Still is http1 . Enable http 2, add to listen directive **http2**.To test this run : curl –Ik <https://163.172.171.154> => http1.1 is obtained! When running in browser same ip with / http2.0 is obtained in Mozila, but when using some /index.html then http1.1 is obtained ????

## Server Push

From nginx 1.13.9 (20/2/2018)support for http/2 server push.

<https://www.nginx.com/blog/nginx-1-13-9-http2-server-push/>

When requesting /index.html = >can be seen that after fetching index.html simultaneously .css and .js are obtained(multiplexed stream). What we can do is to fetch them along with .html file. You can see that in browser or nghttp2 in linux.

Use nghttp2 : **$ apt-get install nghttp2-client**

<https://nghttp2.org/>

**$ nghttp –nys https://163.172.171.154/index.html** (n flag -> is to discart responses only testing them not ssaving on disk,y -> ignore self assigned certificate, s-> print response statistics).

Needed to download latest version 1.34. there was no –y option.

Run with –a flag (to request all assets connected with this request and we should see .css and .js files also with .html): **$ nghttp –nysa https://163.172.171.154/index.**

In order to have resources linked with .html we h ave to add http2\_push /stlle.css (request to the resources, not resource itself).

Add again **$ nghttp –nys** [**https://163.172.171.154/index**](https://163.172.171.154/index)we will obtain all 3 resposnes and asterix (\*) indicating a push response. A much faster and efficient way of delivering web based content

## HTTPS (ssl)

When having http request on port 80 we obtain an error (http://163.), we have to deal with insecure http request, server doesn’t have any listen directive on port 80, but <https://163> is working

To solve:

* One option is to make server available on port 80 over insecure connection (https is a standard and there is no legitimate reason to use http any longer).
* Second option is to redirect (with 301 status code) all http request (from port 80) to equivalent https request(**https:443**).

Most reliable way to achieve this is to create dedicated virtual host or server context.

*server {*

*Listen 80;*

*Server\_name 163.172/171/165;*

*Return 301* [*https://$host$request\_uri*](https://$host$request_uri)*;}*

Test it :***curl –Ik*** [***http://serverip/***](http://serverip/) *=> Http/1.1 301 Moved Permanently -> location* [*https://host/*](https://host/)

* To disable server tokens (not visible when having 404 or in cURL 201 use directive **server\_tokens off**)
* To enable IPv6 or IPng(next generation) (has infinite number of addresses(ipv4 4.3 million), ping6, format hex, 128 bits (ipv4 only 32) in IP address) use directive **listen [::]:443 ssl https;**
* In server context disable ssl protocol (secure socket layer) although we are using it in location to replace it with tls (Transport Layer Security). To enable it use directive: **ssl\_protocols TLSv1 TLSv1.1 TLSv1.2;**
* Optimize cipher suits to encrypt our connection:

**ssl\_prefer\_server\_ciphers on; # we can set which suits to use and which not to use with !**

**#Suite combinations**

**ssl\_ciphers ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH+3DES:!ADH:!AECDH:!MD5;**

**#Enabel Diffie helman key exchange (DH parameter, exchange keys client-server with perfect secrecy)**

**ssl\_dhparam /etc/nginx/ssl\_anel/dhparam.pem;**

**# Create dhparam (size must match private key 2048:**

**openssl dhparam 2048 –out /etc/nginx/ssl\_anel/dhparam.pem**

**#Enable HSTS (http strict transport security) –**header which tells the browser not to load anything over http, from 80 to 443. This blocks access to the pages or subdomains that can only be served over http.

**Add\_header Strict-**Transport-Security “max-age=3153600; includeSubDomains” always;

**# Enable simple cache for SSL sessions** (set handshakes between client and sever **).**

Default type cache is **“builtin**” but is limited to a specific worker process.

Type shared is cached in memory and can be access by any worker process.

In order to use this session cache without actually having the server access the cache we can enable **ssl\_session\_tickets** which means provide the browser with a ticket which validates SSL session. This ticket is issued by the server, it is trusted and allows us to bypass reading from the session cache.

**ssl\_session\_cache shared:SSL:40m;**

**ssl\_session\_timeout 4h;**

**ssl\_session\_tickets on;**

<https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange>

<https://hackernoon.com/algorithms-explained-diffie-hellman-1034210d5100>

## Rate limiting module

Use **s\_client** to see details of connections (TLS for example):

**$ openssl s\_client –connect 163.172.154:443**

It is nginx built in module. It can be used to create traffic light for incoming connections.

Rate limiting module implies not only limiting but rather managing incoming conncetions to the server for a specific reason.

Reasons:

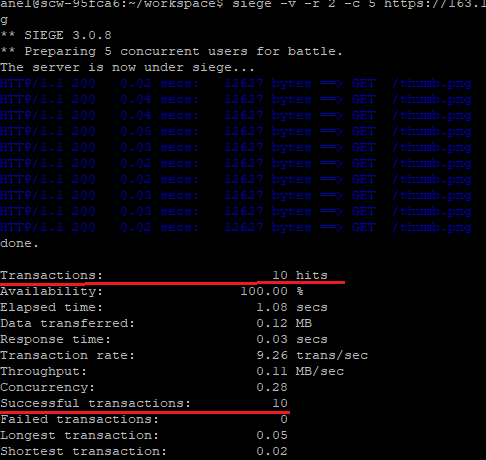
* Security (Brute Force Protection),
* Reliability (Prevent Traffic Spikes),
* Shaping( Service Priority).

Install new tool called **SIEGE**. It is focused more on load testing the server rather than benchmarking as apache benchmark.

<https://www.joedog.org/siege-home/>

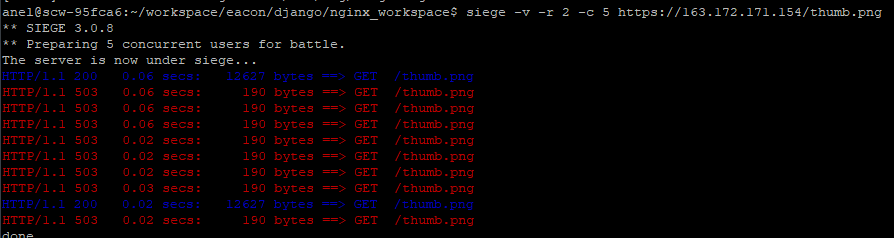
**$apt-get install siege**

To test 10 requests (repetitions x cuncourrent user, can have a delay in seconds –d10 between each user request) run: **siege –v –r 2 –c 5** [**https://163.172.171.154/thumb.png**](https://163.172.171.154/thumb.png)



Like fastcgi cache and ssl session cache, we will define new memory zone in which to track connection limits.

* #Define limit zone: First parameteris a key with wich to identify how rate limiting is applied. Can be $server\_name , $binary\_remote\_address, $request\_uri. If server\_name => rate limiting will apply on requests based on server name (all requests to server\_name IP).If $binary\_ (connecting IP address) => rate limiting will be applied per user (rate limiting a log in form, slowly down brute force attack).If some request\_uri receives more connections requests then the zone allows apply rate limiting. Next define zone name and size of 10MB. 60r/m == 1r/s. Point is that nginx will apply rate limits evenly thus preventing traffic spikes. We can add it in server context or apply to particular location context. To call it use limit\_req zone=MYZONE
* Directive **limit\_req\_zone** **$request \_uri zone=MYZONE:10m rate=60r/m; #or 1r/s**



What this does is that server receives 5 connections at once (first batch of siege request), it responded to the first one of them and rejected remaining 4. In next batch (-r 2) 1 second didn’t finish and again responds to 1 while others are rejected.

* We can add at the end **burst allowance** to all implementation of MYZONE **burst=5;**  OR to add it in location . This means our server will accept 1r + 5 within a second
* When calling siege –v –r 2 –c 5 … = > without burst 1 will be accepted all 9 others will be rejected with 503 error. With burst it means that instead of accepting 1 request and rejecting 4 of them, we will let 1+5 of them to be fullfiled within a second. It doesn’t mean that these 5 will be responded to immediately, but they will still have to adhere to specified limit. Traffic shaping
* Add **nodelay** to control traffic in conjuction with burst.
* <https://www.nginx.com/blog/rate-limiting-nginx/> => **limit\_req\_log\_level warn** #instead of error generate warning for logs I **limit\_req\_status 444** # different than 503(which is default)
* White list and black list (geo and map)
* Deny all requests for specific url: **deny all;**

<https://medium.freecodecamp.org/nginx-rate-limiting-in-a-nutshell-128fe9e0126c>

<https://www.nginx.com/blog/rate-limiting-nginx/>

## Basic Auth

By means of rate limiting we have seen how to slow down brute force attacks and denial of service attacks.

If we have area of site not intended for users but rather for admins or contributors of the site /admin/ where you want to deny any unauthorized request.

Basic auth provoides username and password layer to any part of your site and is relatively straightforward to implement.

Generate simple password file in .htpasswd format.

**$ apt –get install apache2-utils**

**$ htpasswd –c /etc/nginx/.htpasswd anel** (**c** flag indicates that we want to generate password written in a file, for a given username : anel).We will be prompted for a password. The file has ***anel:hashvalueofpassword***

In location/ add directive **auth\_basic “Secure Area”; auth\_basic\_user\_file /etc/nginx/.htpasswd;**

Block of secure area of your site !

## Hardening NGINX

## Additional problems that occurred:

In order to start nginx use systemctl : **sudo systemctl start nginx && ps aux|grep nginx**

If there is an error in your nginx.conf file then: **sudo systemctl reload nginx will fail**

|  |
| --- |
| *root 10658 0.0 0.0 45408 2016 ? Ss 15:21 0:00 nginx: master process /usr/sbin/nginx -g daemon on; master\_process on;*  *www-data 10659 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process*  *www-data 10660 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process*  *www-data 10661 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process*  *www-data 10662 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process*  *www-data 10663 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process*  *www-data 10664 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process*  *www-data 10665 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process*  *www-data 10666 0.0 0.0 45776 3004 ? S 15:21 0:00 nginx: worker process* |

User that is worker is *www-data* , because that is specified in */etc/nginx/nginx.conf* file (user *www-data*)

For temporary enabling/disabling services : **sudo service apache2 stop**

With systemd we can do the following: **systemclt start apache2**

**401 – unauthorized status code**

Still the same version:

sudo nginx -v

nginx version: nginx/1.10.3 (Ubuntu)

Not 1.15.5 as expected !

/usr/bin/nginx -V

nginx version: nginx/1.15.5

$ whereis nginx

/usr/bin/nginx.old

/usr/bin/nginx

/usr/sbin/nginx (this is problematic)

/usr/local/nginx/

/usr/share/nginx/

/etc/nginx/

$ type nginx

nginx is hashed (/usr/sbin/nginx)

$ which nginx

/usr/sbin/nginx

PATH is used to find executables, shell needs to search the path. Shell keep list of programs that has already found – hash

$ hash –l # to list the hashes

$ hash –r # to reset the hashes

$ hash –l # list again

To list the year of files use $ ls --full-time

Echo $PATH

How to add to $PATH = > export $PATH ????

To see the number of commits **: $ git rev-list –-all –-count**

To see the IP address of a hostname **: $ getent hosts eacon.ba**

curl -I <https://185.99.1.225>

Error page !

|  |
| --- |
| limit\_req\_log\_level error; limit\_req\_status 429; error\_page 429 /custom\_429.html; server { listen 80;  location = /custom\_429.html { root /usr/share/nginx/html;  }  location = /image.png { root /usr/share/nginx/html/429x; }  reference the image in custom\_429.html with "src=image.png" and store image.png in /usr/share/nginx/html/429x. Then it should deliver image. |